TARDEC

---TECHNICAL REPORT---

THE NATION'S LABORATORY FOR ADVANCED AUTOMOTIVE TECHNOLOGY

WINNER OF THE 1994 FEDERAL QUALITY IMPROVEMENT PROTOTYPE AWARD

No. 13811

1994 Quality Improvement Prototype Award

TARDEC

Real-Time Powertrain Module for Vehicle Simulation

By W. Bylsma

U.S. Army Tank-Automotive Research,
Development, and Engineering Center
Detroit Arsenal
Warren, Michigan 48397-5000

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (FROM - TO)
01-04-2002		xx-03-2002 to xx-04-2002

4. TITLE AND SUBTITLE Real-Time Powertrain Module for Vehicle Simulation Unclassified	5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Bylsma, Wesley ;	5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME AND ADDRESS U.S. Army Tank-automotive and Armaments Command National Automotive Center ATTN: AMSTA-TR/MS157 Warren, MI48397-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME AND ADDRESS ,	10. SPONSOR/MONITOR'S ACRONYM(S) 11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APUBLIC RELEASE
,

13. SUPPLEMENTARY NOTES

14. ABSTRACT
A two state, engine speed and wheel speed, powertrain model is developed with bondgraph theory and coded in the C programming language for realtime simulation and comparison to the same model implemented in the MATLAB Simulink/Stateflow environment. Both throttle and braking are inputs with adjustable engine, transmission, and vehicle parameters. The structure of the model implementation provides for decoupled transmission and driveline modules for modularity. Gear shifting is accomplished with a shift delay of 0.8 seconds. A fixed step Runge-Kutta integration scheme is used with a time step of 0.01 seconds. Two throttle input cases are compared for each model with the results being essentially the same for each case - peak differences are attributed to signal shift on transition edges. Simulation time for each case was 200 seconds and was achieved within 5 seconds realtime. This is a 40:1 simulation time to real-time ratio.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19. NAME OF RESPONSIBLE PERSON
			Public Release	65	Fenster, Lynn lfenster@dtic.mil
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER International Area Code Area Code Telephone Number 703767-9007 DSN 427-9007

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	APRIL 2002	MARCH - APRIL 2002

4. TITLE AND SUBTITLE
REAL-TIME POWERTRAIN MODULE FOR VEHICLE SIMULATION

5. FUNDING NUMBERS

6. AUTHOR(S)
Wesley Bylsma

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Tank-automotive and
Armaments Command/National
Automotive Center
ATTN: AMSTA-TR-N/MS157
Warren, MI 48397-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
13811

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release: Distribution is unlimited	A

13. ABSTRACT *(Maximum 200 Words)*

A two state, engine speed and wheel speed, powertrain model is developed with bondgraph theory and coded in the C programming language for real-time simulation and comparison to the same model implemented in the MATLAB Simulink/Stateflow environment. Both throttle and braking are inputs with adjustable engine, transmission, and vehicle parameters. The structure of the model implementation provides for decoupled transmission and driveline modules for modularity. Gear shifting is accomplished with a shift delay of 0.8 seconds. A fixed step Runge-Kutta integration scheme is used with a time step of 0.01 seconds. Two throttle input cases are compared for each model with the results being essentially the same for each case - peak differences are attributed to signal shift on transition edges. Simulation time for each case was 200 seconds and was achieved within 5 seconds real-time. This is a 40:1 simulation time to real-time ratio.

14. SUBJECT TERMS	15. NUMBER OF PAGES
real-time, powertrain, automatic transmission, simulation	62
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unclassified

Table of Contents

Technical Report 13811
April 2002

Real-Time Powertrain Module for Vehicle Simulation

W. Bylsma
U.S. Army Tank-automotive and Armaments Command
Research, Development and Engineering Center
National Automotive Center
Warren, Michigan 48397-5000

ABSTRACT
A two state, engine speed and wheel speed, powertrain model is developed with bondgraph theory and coded in the C programming language for real-time simulation and comparison to the same model implemented in the MATLAB Simulink/Stateflow environment. Both throttle and braking are inputs with adjustable engine, transmission, and vehicle parameters. The structure of the model implementation provides for decoupled transmission and driveline modules for modularity. Gear shifting is accomplished with a shift delay of 0.8 seconds. A fixed step Runge-Kutta integration scheme is used with a time step of 0.01 seconds. Two throttle input cases are compared for each model with the results being essentially the same for each case - peak differences are attributed to signal shift on transition edges. Simulation time for each case was 200 seconds and was achieved within 5 seconds real-time. This is a 40:1 simulation time to real-time ratio.

INTRODUCTION
Advances in the analytical world of modeling and simulation can be attributed to not only faster microprocessors and computing power that speed up simulation times, but new formulations of mathematical models that in and of themselves result in quicker processing speeds. This capability has opened up the possibility for new approaches to modeling and simulation to be explored. One of these is real-time simulation. No longer is the vehicle manufacturer limited to evaluating their vehicle on the computer screen, but can experience their design in the physical world through the use of motion simulators. This "human-in-the-loop" or "hardware-in-the-loop" (HIL) approach requires the models driving the motion simulators to perform in real-time for the most accurate and realistic representation of how the vehicle will perform once manufactured.

This naturally leads to the definition of what really does "real-time" mean. In general, real-time is achieved if the tasks required to be performed are done so within a specified amount of time. For vehicle dynamics a time step of around 1 millisecond will provide most of the frequency content needed for a representative model. This of course depends upon what specific vehicle attribute is being looked at and might require more refinement.

This report addresses the need for a real-time powertrain model to be used with any vehicle model on a motion simulator. In particular, it consists of an engine, automatic transmission, and driveline. The driveline has been separated out so other configurations can be accommodated. The torque converter's operation is based on the speed ratio between the engine and driveline at the converter which provides the capacity factor (K) and torque ratio of the transmission. The assumption of no compliance in the driveline and no losses were made to compare this model with a MATLAB Simulink/Stateflow demonstration model of an automatic transmission upon which it is based. Because the model equations were converted into a bondgraph the compliances and losses can easily be added into the bondgraph and the model equations regenerated. The actual real-time powertrain model was generated based on the bondgraph and formulated in C code.

MODEL DESCRIPTION

The model is a simplified automatic transmission consisting of an engine, torque converter, gear shift mechanism, transmission, and vehicle. The vehicle characteristics are lumped together to allow implementation of the powertrain model without focusing on the details of the vehicle dynamics. Figure 1 shows the general structure. This is taken from a MATLAB Simulink/Stateflow demonstration so that the resulting bondgraph and simulation results can be compared.

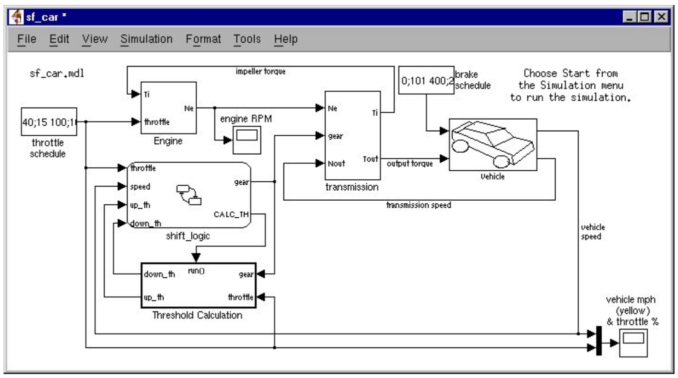


Figure 1 - MATLAB Simulink/Stateflow Diagram

The engine receives a throttle command which is its input. The engine torque is obtained from an engine map which gives an engine torque for a specific throttle setting and engine speed. (See Appendix B for graphs and tables of these values.)

$$I_{ei}\dot{N}_e = T_e - T_i$$
$$I_{ei} = engine + impeller\ moment\ of\ inertia$$
$$N_e = engine\ speed$$
$$T_e = f(throttle, N_e) = engine\ torque$$
$$T_i = impeller\ torque$$

The engine is connected to the impeller of the torque converter which provides the mechanism for putting torque into the transmission.

$$T_i = \left(\frac{N_e}{K}\right)^2 impeller\ torque$$
$$K = f(N_{in}/N_e) = capacity\ or\ K-factor$$
$$N_{in} = turbine\ speed = transmission\ input\ speed$$
$$T_t = R_{tq}T_i = turbine\ torque$$
$$R_{tq} = f(N_{in}/N_e) = torque\ ratio$$

The torque characteristics of the torque converter are determined from the engine speed (impeller input) and turbine (output) speed. A lookup table defines the capacity factor and torque ratio based on the speed ratio. The transmission itself is modeled as a manual transmission with small shift times to emulate the automatic shifting.

$$R_{tr} = f(gear) = transmission\ ratio$$
$$T_{out} = R_{tr}T_{in}$$
$$N_{in} = R_{tr}N_{out}$$
$$T_{in}, T_{out} = transmission\ input\ and\ output\ torque$$
$$N_{in}, N_{out} = transmission\ input\ and\ output\ speed$$

The transmission gear ratio depends on the gear state. The current model uses only four gears--this could be adjusted easily. The output of the transmission is connected through a final drive to a simplified "wheel" system. The whole vehicle inertia is accounted for in one wheel. This gives the necessary states to the powertrain model without involving the details of various drivelines (differentials, all-wheel-drive, six wheel and eight wheel configurations, etc.).

$$I_v\dot{N}_w = R_{fd}T_{out} - T_{load}$$
$$I_v = vehicle\ inertia$$
$$N_w = wheel\ speed$$
$$R_{fd} = final\ drive\ ratio$$
$$T_{load} = f(N_w) = load\ torque$$

The load torque includes all the running gear forces and vehicle resistances as well as the braking torque.

$$T_{load} = \mathrm{sgn}(mph)(R_{load0} + R_{load2}mph^2 + T_{brake})$$
$$T_{load} = load\ torque$$
$$R_{load0} = friction\ coefficient$$
$$R_{load2} = aerodynamic drag\ coefficient$$
$$T_{brake} = brake\ torque$$
$$mph = vehicle\ linear\ velocity$$

This completes the continuous portion of the powertrain model. The gear shifting composes the discrete part. A lookup table is used to determine the down and up shift speeds based on the current gear and throttle input. If either of these conditions are met, a flag is set to indicate

entering the down or up shift mode and the current time is recorded. Once in this mode, if the speed (up or down shift) condition is not met before the shift delay time has been completed, the flag is reset and steady state operation is resumed. Once the shift delay time has been satisfied and the shift (up or down) flag is still set, the gear state is changed to the new gear (up or down) and the flag reset. The MATLAB Stateflow diagram for this logic is included in Appendix A. All model units are in feet, pounds force, and seconds.

MODEL IMPLEMENTATION

Bondgraph theory was used to develop a model of the powertrain as shown in Figure 2.

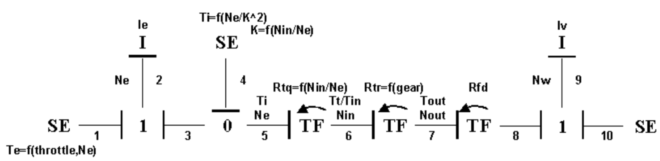


Figure 2 - Bondgraph Model

This allows efficient development of the state equations for implementing the model with a general programming language in an environment that will run real-time. From the bondgraph in Figure 2 the equations for the powertrain model were derived (see Appendix C) and were coded in the C programming language. The C language offers both portability and speed. Implementation in other programming languages are possible as well. The two states are Ne (p2dot) and Nw (p9dot)

$$p2dot = e2 = Te - Ti$$
$$p9dot = e9 = Rfd*Rtr*Rtq*Ti - Tload.$$

A modular structure was preserved to allow the engine/transmission to interface with different driveline configurations. The current implementation can be modified as desired. To allow this modularity to work, the state variables must be available to each module since their functionality is dependent upon each. The two state structures shown below include all necessary variables to calculate the state.

```
{
  /* engine data */
  double p2;   /* engine RPM */
  double cet[RK_ORDER];   /* engine Torque */

  /* torque converter data */
  double sr[RK_ORDER];   /* speed ratio */
  double ck[RK_ORDER];   /* capacity factor */
  double crtq[RK_ORDER];   /* torque ratio */
  double Ti[RK_ORDER];   /* impeller torque */

  /* transmission data */
  double gearstate;   /* gear */
  double crtr;   /* transmission ratio */
  double tdn;   /* time delay for downshift */
  double tup;   /* time delay for upshift */
  int flgd;   /* flag indicator in downshift mode */
  int flgu;   /* flag indicator in upshift mode */

  /* driver inputs */
  double cthrottle[RK_ORDER];   /* throttle input */

  /* temp variables */
  double down_threshold;
  double up_threshold;

} pwr_state;

struct
{
  /* vehicle data */
  double p9;   /* wheel RPM */
  double Tload[RK_ORDER];   /* Load Torque */
  double vspd[RK_ORDER];   /* vehicle speed */

  /* driver inputs */
  double cbrake[RK_ORDER];   /* brake input */

} drvl_state;
```

The brake input is included in the driveline state since its origination is from the wheel spindle. A fourth order fixed step Runge-Kutta integration method is used (h=0.01 sec). This choice reflects the desire for a real-time module to complete updating the state variables within a specified time period. The fixed step attribute of the Runge-Kutta method satisfies that need, although any integration method that meets the real-time requirements can be used instead.

Seven major subroutines make up the computational structure of the powertrain. They are:

`void init_pwr()` - reads in data and initializes the `pwr` and `pwr_state` variables.

`void init_drvl()` - reads in data and initializes the `drvl` and `drvl_state` variables.

`void pwrmod(double t, double h)` - formulates the Runge-Kutta integration variables and updates the states for `pwr_state`.

`void update_pwr_state(double t,int i,double rk)` - update the intermediate Runge-Kutta integration variables for each t and h.

`void drvlmod(double t, double h)` - formulates the Runge-Kutta integration variables and updates the states for `drvl_state`.

`void update_drvl_state(double t,int i,double rk)` - update the intermediate Runge-Kutta integration variables for each t and h.

`void shift_logic(double t,double *up,double *dn)` - determine proper gear state to enter if completed down or up shift time delay.

Several support functions are necessary for the implementation to work. These are:

`int interp1(double *x, double *y, int l, double xi, double *yo)` - one dimensional table lookup with extrapolation.

`int interp2(double *x, double *y, double *z, int m, int n, double xi, double yi, double *zo)` - two dimensional table lookup with extrapolation (uses interp1).

`int readtbl1d(const char *file, int *dim, double arr1d[])` - read in values for a one dimensional table from an input file.

`int readtbl2d(const char *file, int *dim, double arr2d[][dim[2]])` - read in values for a two dimensional table from an input file.

`int printtbl1d(int *dim, double arr1d[])` - print values for a one dimensional table.

`int printtbl2d(int *dim, double arr2d[][dim[2]])` - print values for a two dimensional table.

It should be noted that the shift delay time used was 0.8 seconds. Adjustments to this and other parameters must be made to match the specific engine/transmission being modeled. The source code is contained in Appendix D.

RESULTS

Two cases were considered for comparison. Each contained different percentage throttle inputs as shown in Table 1.

Table 1 - Percentage Throttle Input

	Case 1	Case 2
Time	Throttle (%)	
0.0	60.0	20.0
14.9	40.0	20.0
15.0	100.0	20.0
100.0	0.0	0.0
200.0	0.0	0.0

Appendix E contains the output for each variable of the model. The outputs between the bondgraph (C coded) and MATLAB Simulink/Stateflow were essentially identical. For brevity only the differences in engine RPM and vehicle speed are presented below.

Case 1 Results:

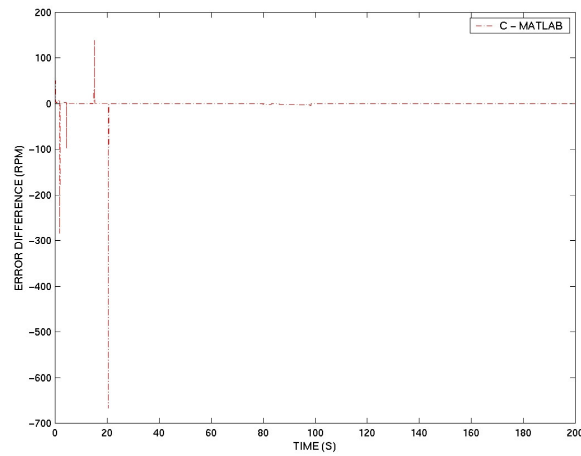


Figure 3 - Case 1 Engine RPM Model Differences

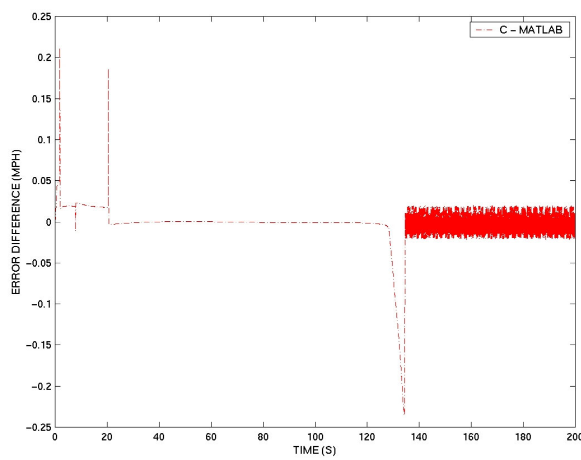


Figure 4 Case 1 Vehicle Speed Model Differences

Case 2 Results:

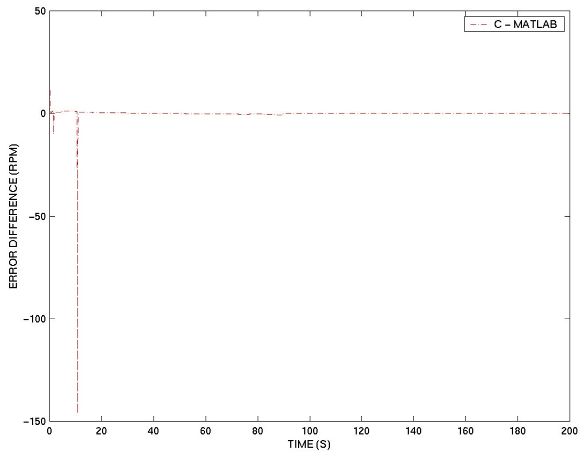


Figure 5 - Case 2 Engine RPM Model Differences

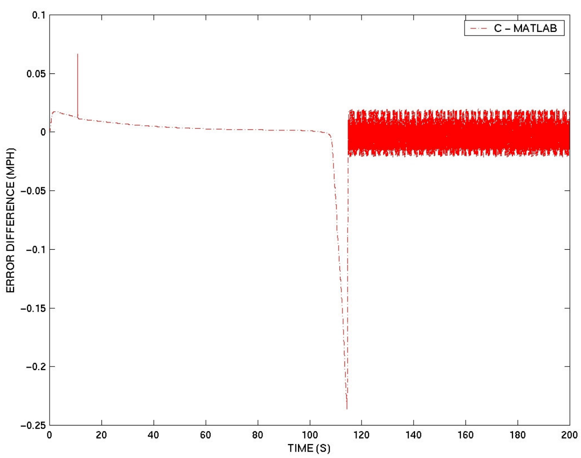


Figure 6 - Case 2 Vehicle Speed Model Differences

The peak values are attributed to minor shift differences in the signal transition edges. The shift delay time can have an effect on this result. The "chatter" in the vehicle speed is attributed to small integration error and signal offset from the integration scheme seeking a constant speed value. To ensure real-time each case was simulated for 200 seconds. Table 2 gives the results.

Table 2 - Simulation Real-Time in seconds.

	Case 1	Case 2
Time	4.24	3.94

This gives about a 40:1 simulation time to real-time ratio. It should be noted that due to processor loads there was some fluctuation in the times reported, but not more than approximately 0.5 seconds. For a worst case scenario it is assumed that the simulation time is 5 seconds. It should also be noted that file and screen output were enabled during each test, thus consuming more time than necessary. The real-time capability reported for this model is therefore on the conservative side.

CONCLUSION

The model conforms to the real-time requirement by a ratio of 40:1. Modifications to the bondgraph to account for resistive losses and compliance in the transmission can be added to provide a more realistic model. An addition of a fuel map would allow computation of vehicle fuel economy if necessary. More complete driveline modules should be developed to provide for specific vehicle configurations (4x4, 6x6, etc.)

CONTACT

The author is a an engineer at the U.S. Army Tank-automotive and Armaments Command, Research, Development and Engineering Center (TACOM-TARDEC). Interested parties can contact the author at the U.S. Army Tank-automotive and Armaments Command, ATTN: AMSTA-TR-N/MS157, Warren, Michigan 48397-5000, bylsmaw@tacom.army.mil.

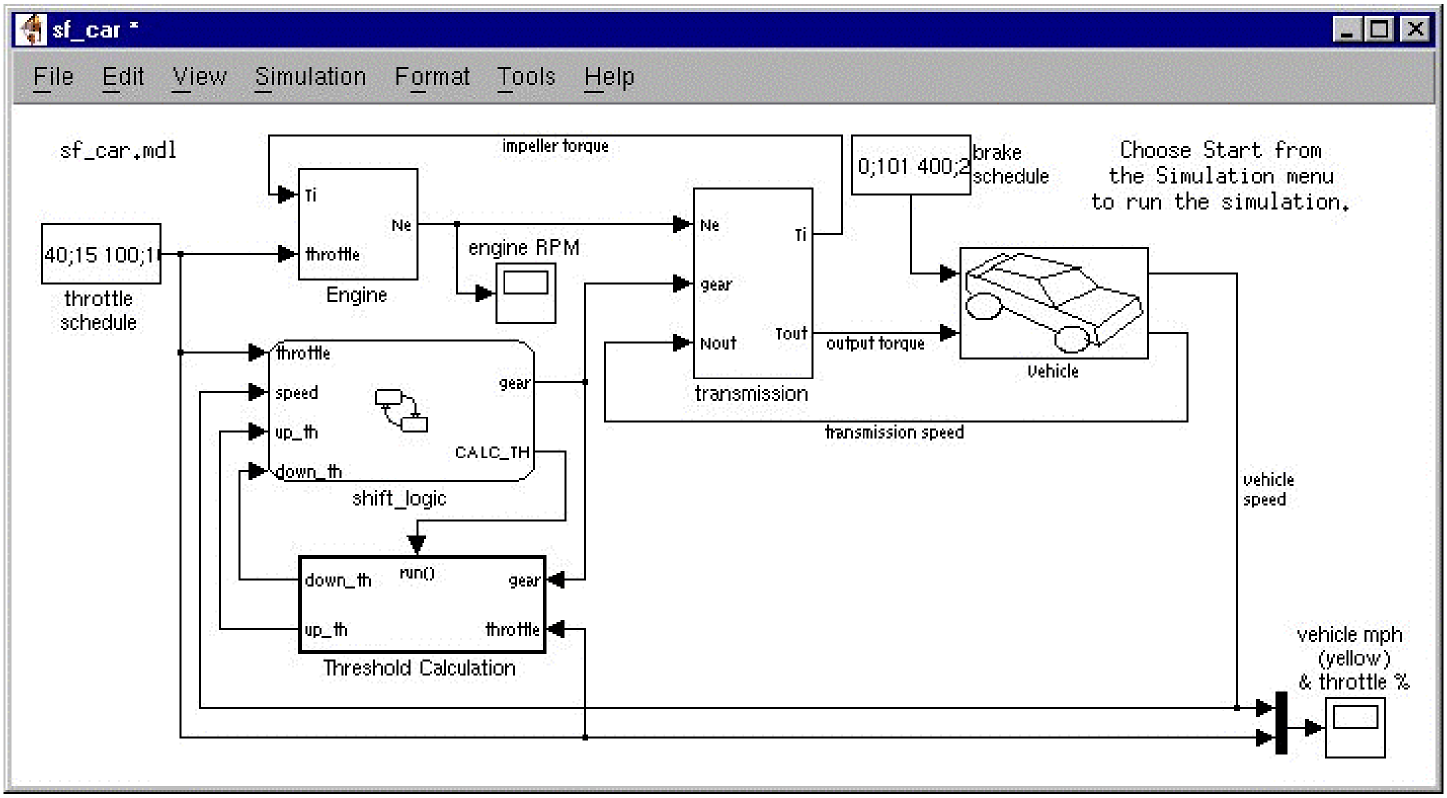
REFERENCES

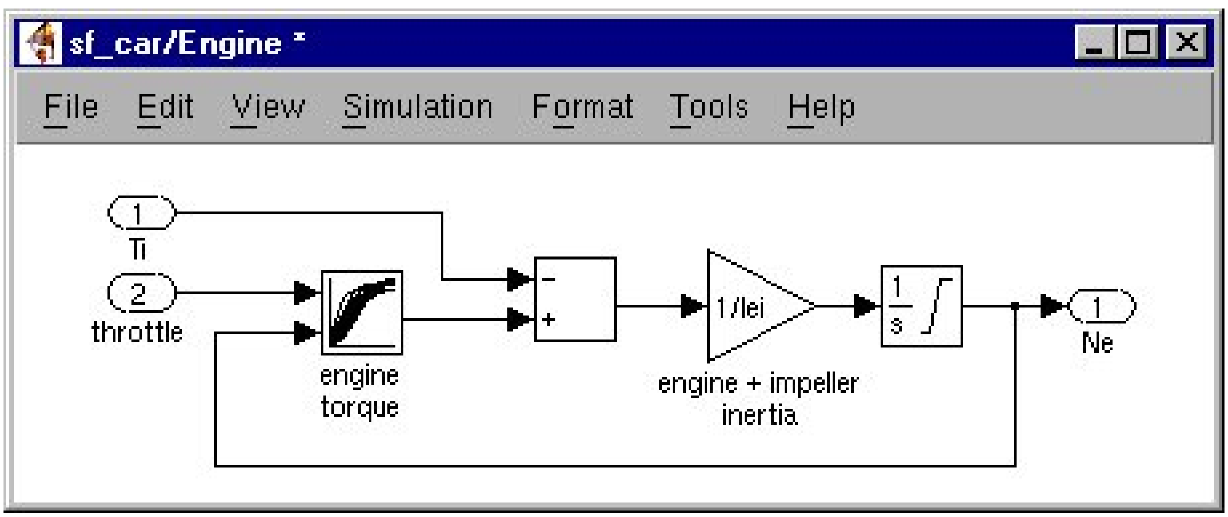
Enabling PC-Based HIL Simulation for Automotive Applications, Paul Baracos, Ph.D., P.Eng , Guillaume Murere, Ph.D., C.A. Rabbath, Ph.D., eng., and Wensi Jin, B.S.(E.E.), Opal-RT, 1751 Richardson, suite 2525, Montreal, QC, Canada H3K 1G6

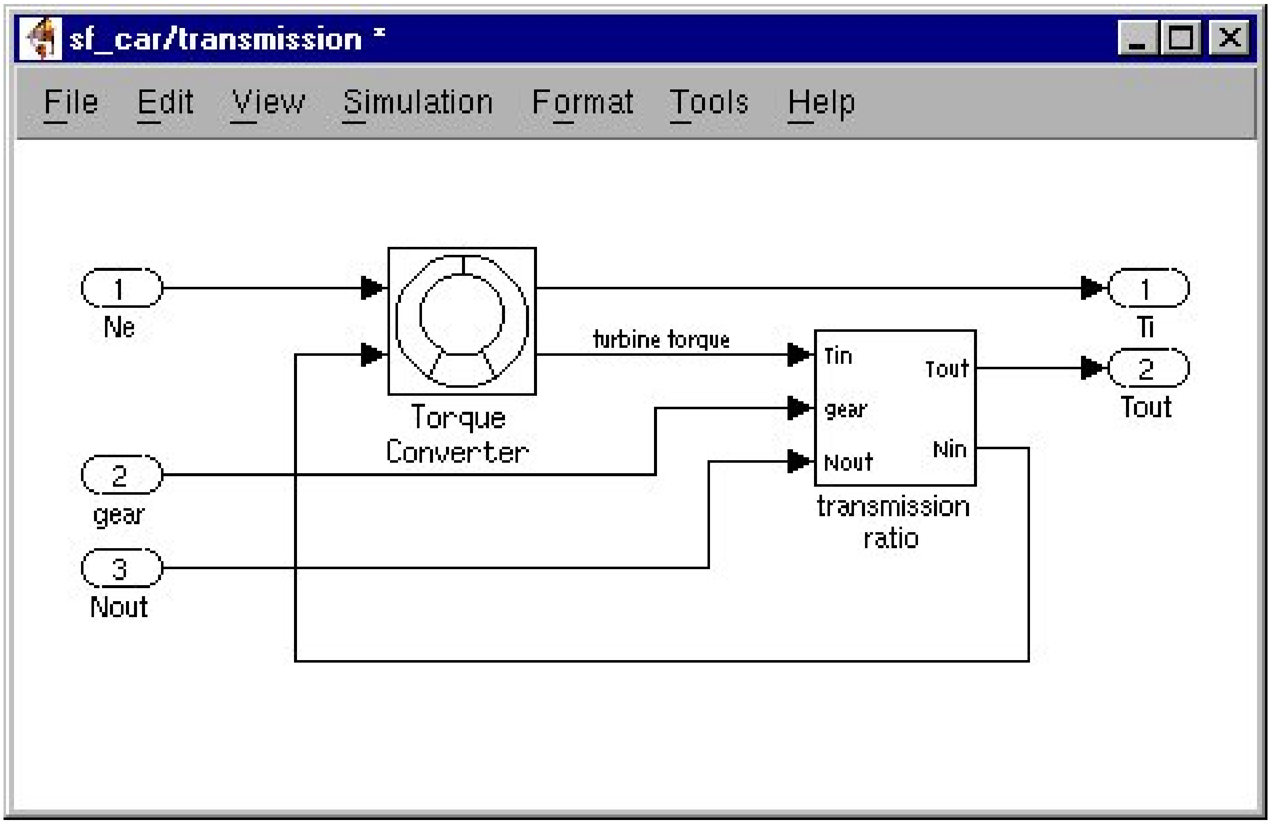
DEFINITIONS, ACRONYMS, ABBREVIATIONS

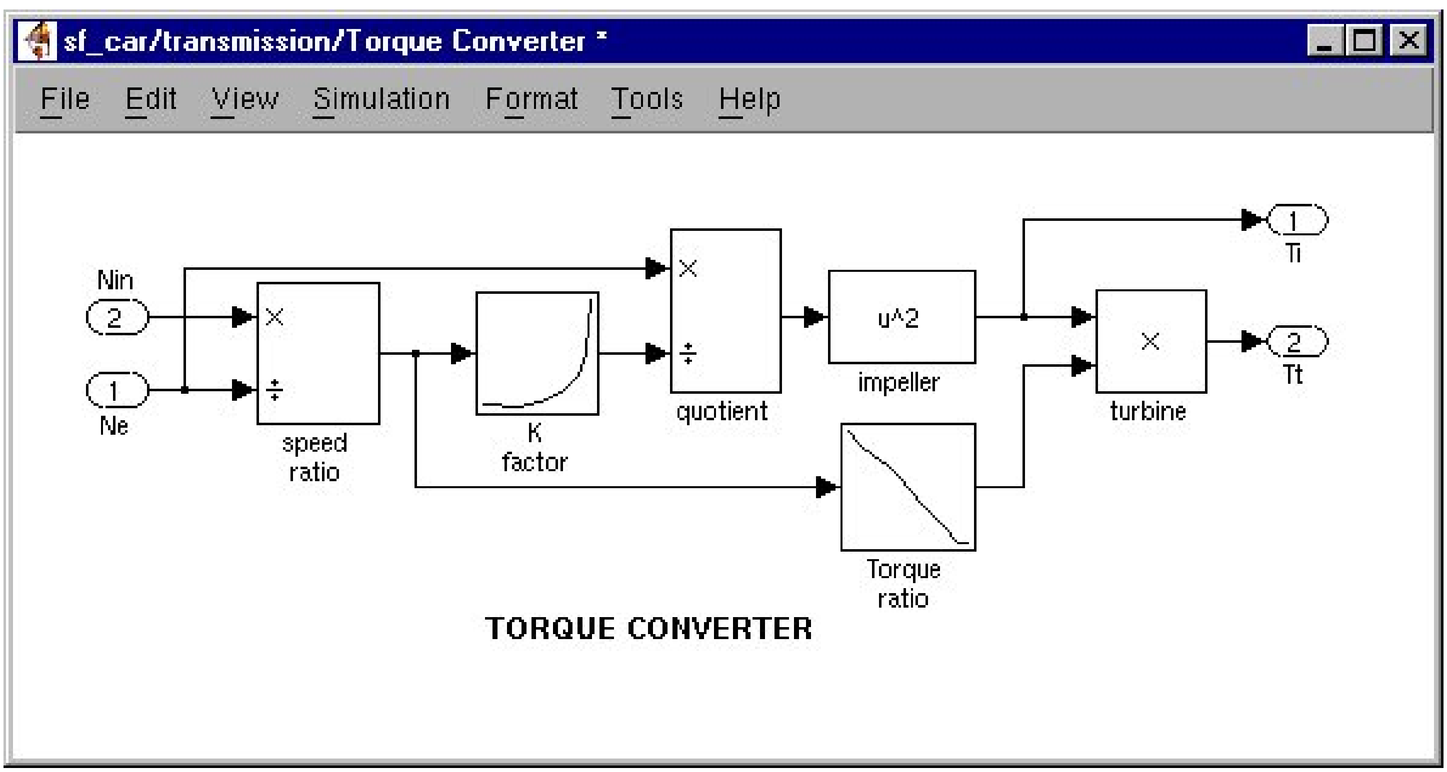
MPH - Mile per hour
TACOM - U.S. Army Tank-automotive and Armaments Command
TARDEC - TACOM Research, Development and Engineering Center
NAC - National Automotive Center

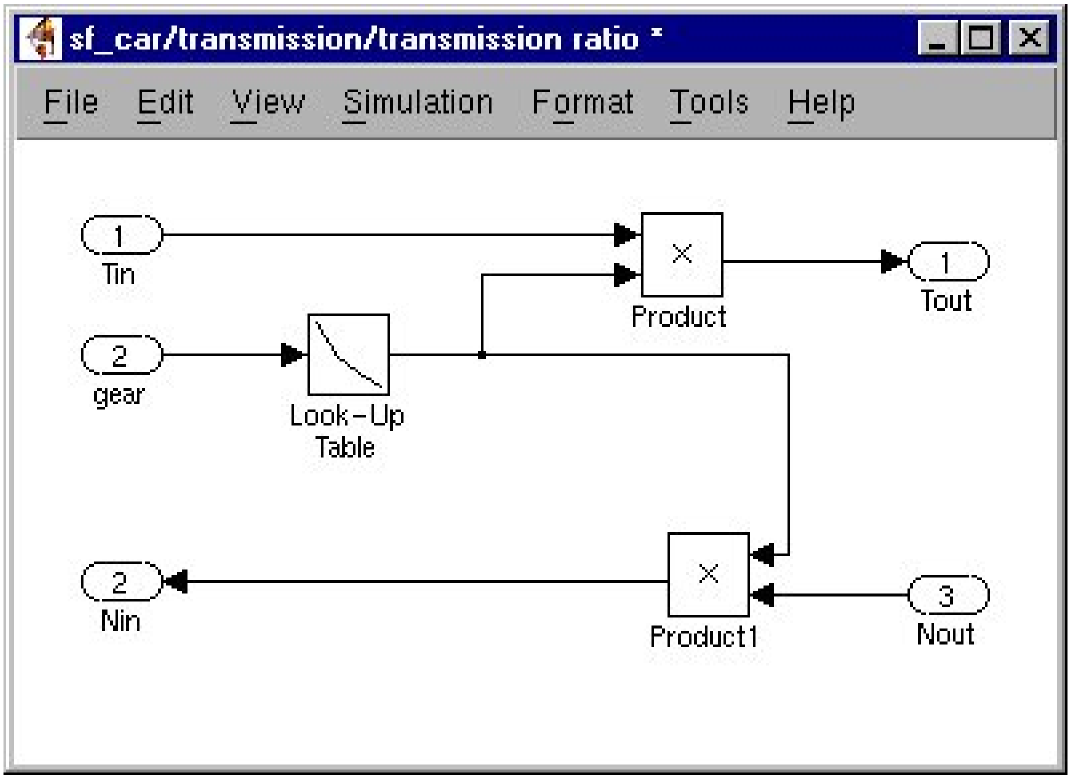
APPENDIX A - MATLAB Simulink/Stateflow Subsystem Diagrams

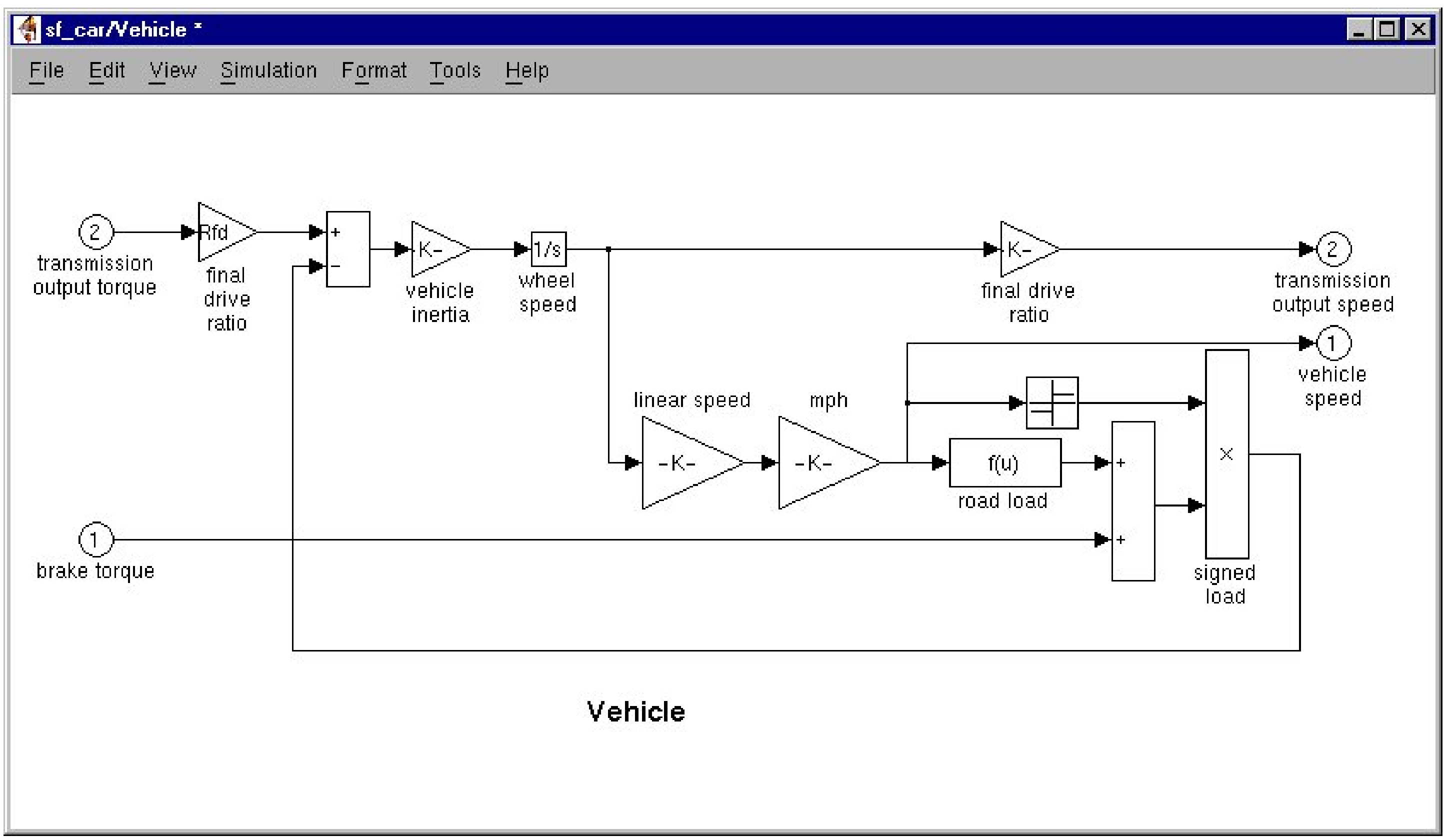


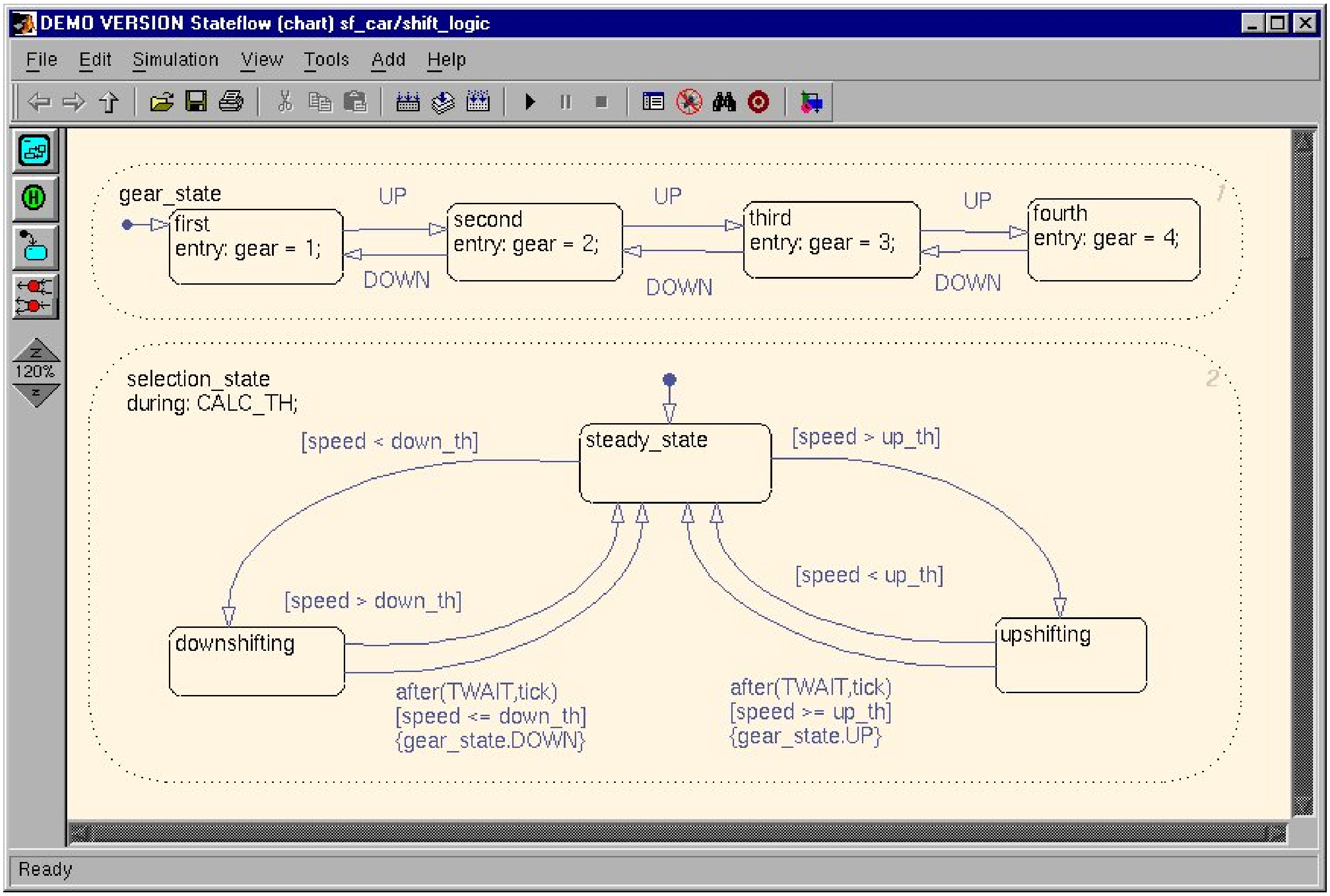


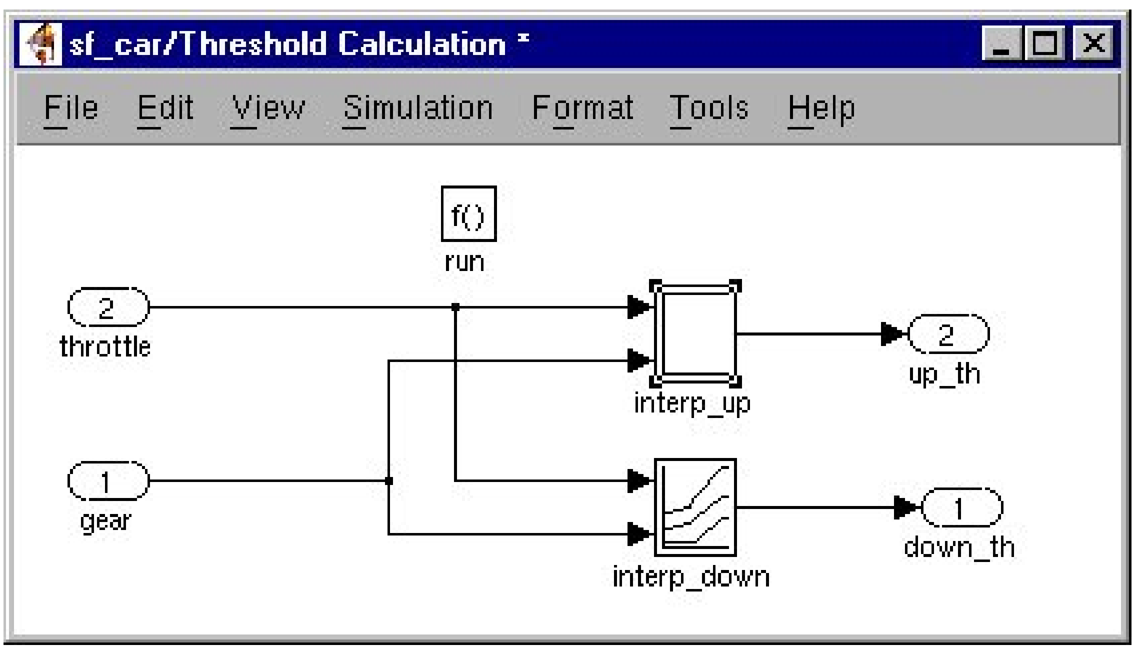

sf_car/transmission *
File Edit View Simulation Format Tools Help
1
Ne
turbine torque
Torque
Converter
2
gear
3
Nout
Tin
gear
Nout
Tout
Nin
transmission
ratio
1
Ti
2
Tout


sf_car/transmission/Torque Converter *
File Edit View Simulation Format Tools Help
Nin
2
1
Ne
speed
ratio
K
factor
quotient
u^2
impeller
Torque
ratio
turbine
1
Ti
2
Tt
TORQUE CONVERTER

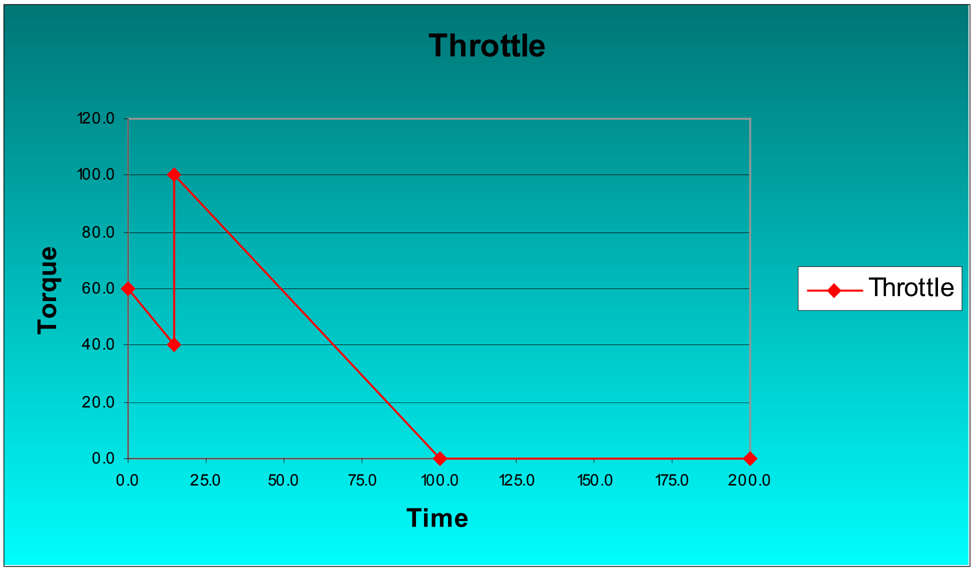

sf_car/transmission/transmission ratio *
File Edit View Simulation Format Tools Help
1
Tin
2
gear
Look-Up Table
Product
1
Tout
2
Nin
Product1
3
Nout


sf_car/Vehicle *
File Edit View Simulation Format Tools Help
2
transmission output torque
Rfd
final drive ratio
vehicle inertia
1/s
wheel speed
final drive ratio
2
transmission output speed
1
vehicle speed
linear speed
mph
f(u)
road load
signed load
1
brake torque
Vehicle

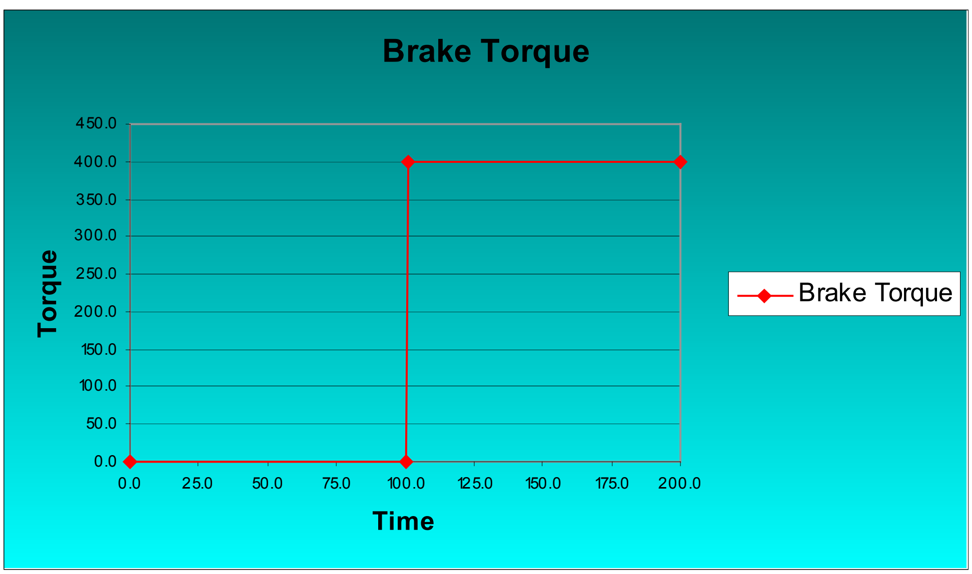

DEMO VERSION Stateflow (chart) sf_car/shift_logic
File Edit Simulation View Tools Add Help
120%
gear_state
first
entry: gear = 1;
UP
DOWN
second
entry: gear = 2;
UP
DOWN
third
entry: gear = 3;
UP
DOWN
fourth
entry: gear = 4;
1
selection_state
during: CALC_TH;
2
steady_state
[speed < down_th]
[speed > up_th]
[speed > down_th]
[speed < up_th]
downshifting
upshifting
after(TWAIT,tick)
[speed <= down_th]
{gear_state.DOWN}
after(TWAIT,tick)
[speed >= up_th]
{gear_state.UP}
Ready


sf_car/Threshold Calculation *
File Edit View Simulation Format Tools Help
f()
run
2
throttle
1
gear
interp_up
interp_down
2
up_th
1
down_th

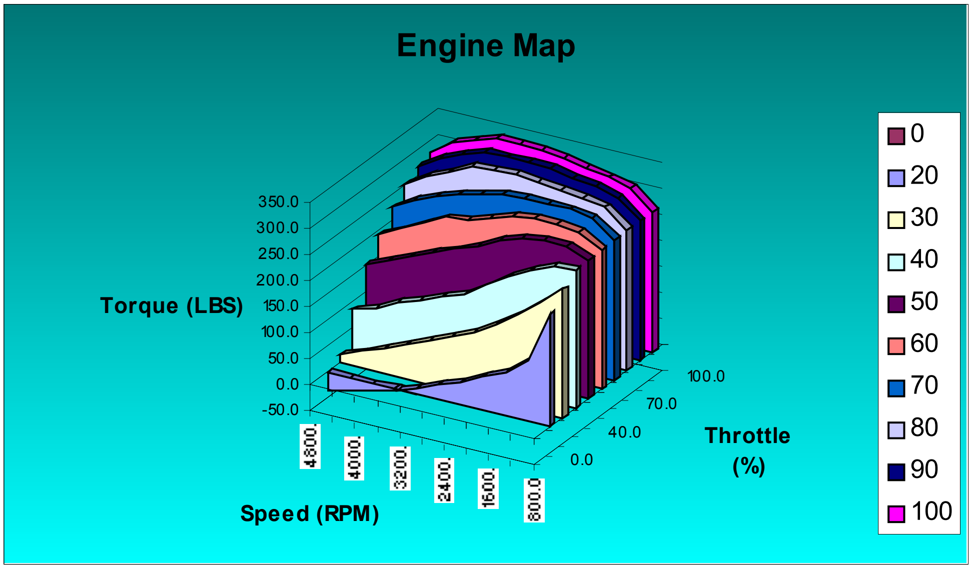
APPENDIX B - Graphs & Tables



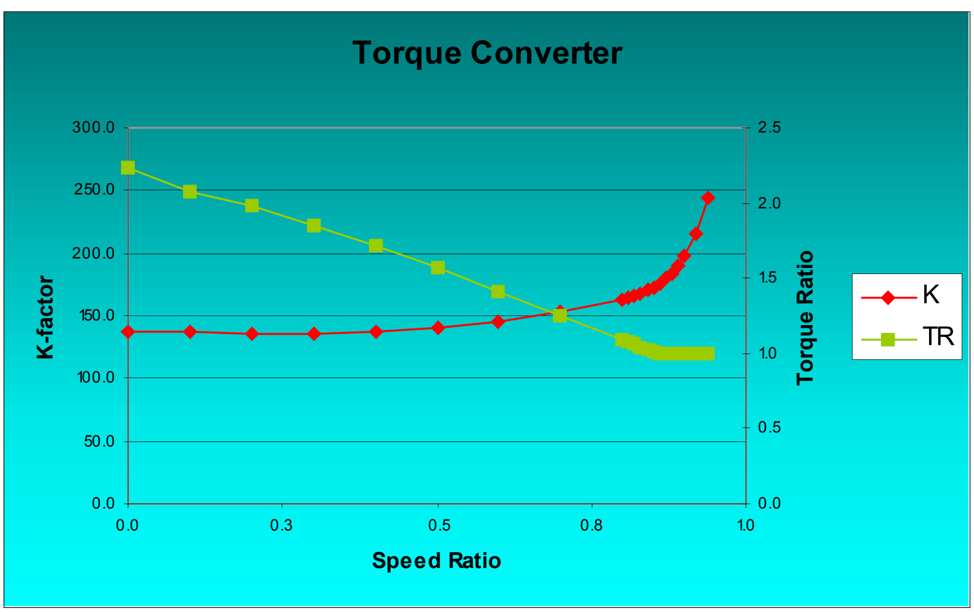
	Case 1	Case 2
Time	Throttle (%)	
0.0	60.0	20.0
14.9	40.0	20.0
15.0	100.0	20.0
100.0	0.0	0.0
200.0	0.0	0.0



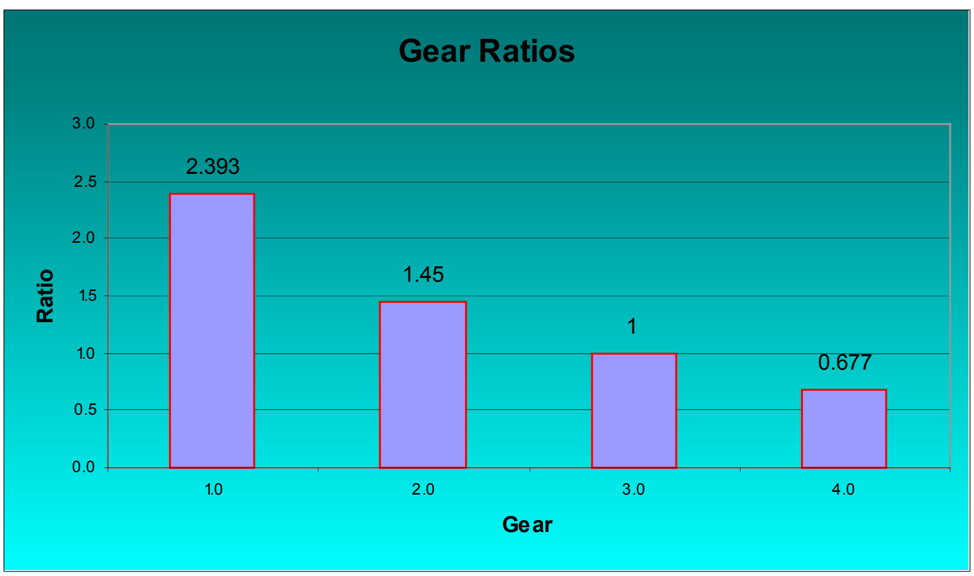
Time	Brake Torque (LBS)
0.0	0.0
100.0	0.0
101.0	400.0
200.0	400.0



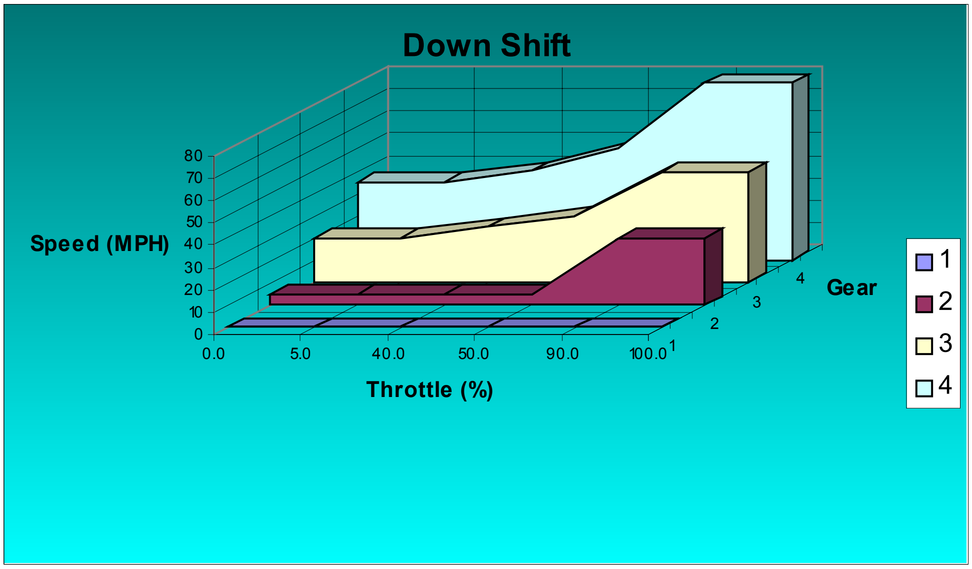
		Engine Speed (RPM)										
		800.0	1200.0	1600.0	2000.0	2400.0	2800.0	3200.0	3600.0	4000.0	4400.0	4800.0
Throttle (%)	0.0	-40.0	-44.0	-49.0	-53.0	-57.0	-61.0	-65.0	-70.0	-74.0	-78.0	-82.0
	20.0	215.0	117.0	85.0	66.0	44.0	29.0	10.0	-2.0	-13.0	-22.0	-32.0
	30.0	245.0	208.0	178.0	148.0	122.0	104.0	85.0	66.0	48.0	33.0	18.0
	40.0	264.0	260.0	241.0	219.0	193.0	167.0	152.0	133.0	119.0	96.0	85.0
	50.0	264.0	279.0	282.0	275.0	260.0	238.0	223.0	208.0	189.0	171.0	152.0
	60.0	267.0	290.0	293.0	297.0	290.0	275.0	260.0	256.0	234.0	212.0	193.0
	70.0	267.0	297.0	305.0	305.0	305.0	301.0	293.0	282.0	267.0	249.0	226.0
	80.0	267.0	301.0	308.0	312.0	319.0	323.0	319.0	316.0	297.0	279.0	253.0
	90.0	267.0	301.0	312.0	319.0	327.0	327.0	327.0	327.0	312.0	293.0	267.0
	100.0	267.0	301.0	312.0	319.0	327.0	334.0	334.0	334.0	319.0	305.0	275.0



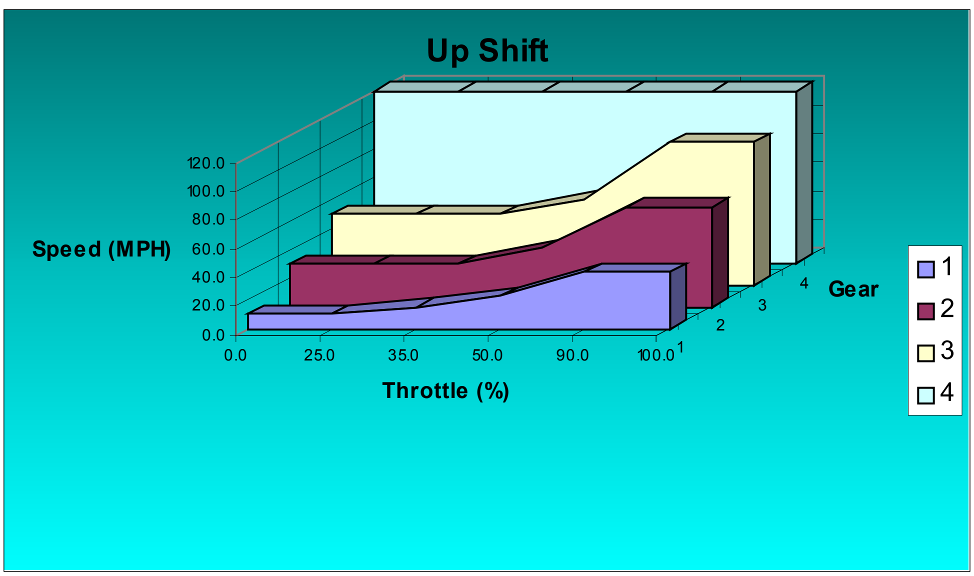
Speed Ratio	K-factor	Torque Ratio
0.0	137.5	2.2
0.1	137.1	2.1
0.2	135.9	2.0
0.3	135.7	1.8
0.4	137.6	1.7
0.5	140.4	1.6
0.6	145.3	1.4
0.7	152.9	1.3
0.8	163.0	1.1
0.8	164.3	1.1
0.8	166.2	1.1
0.8	168.0	1.0
0.8	170.1	1.0
0.9	172.8	1.0
0.9	175.4	1.0
0.9	179.6	1.0
0.9	183.6	1.0
0.9	189.9	1.0
0.9	197.7	1.0
0.9	215.9	1.0
0.9	244.5	1.0



Gear	Ratio
1.00	2.39
2.00	1.45
3.00	1.00
4.00	0.68



		Gear			
		1	2	3	4
Throttle (%)	0.0	0.0	5.0	20.0	35.0
	5.0	0.0	5.0	20.0	35.0
	40.0	0.0	5.0	25.0	40.0
	50.0	0.0	5.0	30.0	50.0
	90.0	0.0	30.0	50.0	80.0
	100.0	0.0	30.0	50.0	80.0



		Gear			
		1	2	3	4
Throttle (%)	0.0	10.0	30.0	50.0	1000000.0
	25.0	10.0	30.0	50.0	1000000.0
	35.0	15.0	30.0	50.0	1000000.0
	50.0	23.0	41.0	60.0	1000000.0
	90.0	40.0	70.0	100.0	1000000.0
	100.0	40.0	70.0	100.0	1000000.0

Engine Inertia	0.022
Final Drive Ratio	3.23
Rolling Resistance	40
Air Drag Coefficient	0.02
Wheel Radius	1
Vehicle Inertia	12.0941

APPENDIX C - Bondgraph Equations

Step 1 - Constitutive Relationships

e1 = Te = f(throttle,Ne)
f2 = p2/Ie
e4 = Ti = f(Ne,K)
f9 = p9/Iv
e10 = Tload = f(vspd,brake)

Step 2 - Junction Equations

e1 - e2 - e3 = 0 -> e2 = e1 - e3
f2 = f3 = f5
e3 = e4 = e5
f5 = Rtq *f6
f6 = Rtr*f7
f7 = Rfd*f8
f8 = f9
e8 - e9 - e10 = 0 -> e9 = e8 - e10

Step 3 - Substitutions

throttle = input
brake = input
f2 = Ne = p2/Ie
f9 = Nw = p9/Iv
Te = interp2(nevec[],thvec[],emap[][],Ne,throttle)
Rtr = interp1(gear[],gearsratio[],gear)
Nin = Rtr*Rfd*Nw
sr=Nin/Ne
K = interp1(sr[],K[],sr)
Ti = (Ne/K)^2
brake = input
vspd = Nw*Rw*2*PI*60/5280 (RPM -> MPH)
Tload = sign(vspd)*[Rload0+Rload2(vspd)^2 + brake]

e3 = e4 = Ti

e5=e6/Rtq
e6=e7/Rtr
e7=e8/Rfd
e8 = Rfd*Rtr*Rtq*e5 = Rfd*Rtr*Rtq*e4 = Rfd*Rtr*Rtq*Ti

Step 4 - State Equations

$$Q = \int f dt, e = KQ$$

$$P = \int e dt, f = P / m$$

$$\dot{Q} = f$$

$$\dot{P} = e$$

f2 = p2/m
p2dot = e2 = Te-Ti
p9dpt = e9 = Rfd*Rtr*Rtq*Ti - Tload

APPENDIX D - C Program Listings

Complile Commands
```
cd src/
cc -c interp1.c
cc -c interp2.c
cc -c readtbl1d.c
cc -c printtbl1d.c
cc -c readtbl2d.c
cc -c printtbl2d.c
cd ..
cc pwr-RT.c -lm src/interp1.o src/interp2.o src/readtbl1d.o src/printtbl1d.o src/readtbl2d.o src/printtbl2d.o
```

Program Listing

```
/* pwr-RT.c */

#include <stdio.h>
#include <math.h> /* atan,pow */

#define PI 4.0*atan(1.0)

#define MAX_CONVERTER_DATA_X 21  /* SR, K, TR */
#define MAX_CONVERTER_DATA_Y 3
#define MAX_DOWNTAB_X 4
#define MAX_DOWNTAB_Y 6
#define MAX_DOWNTH 6
#define MAX_EMAP_X 11
#define MAX_EMAP_Y 10
#define MAX_NEVEC 11
#define MAX_THVEC 10
#define MAX_UPTAB_X 4
#define MAX_UPTAB_Y 6
#define MAX_UPTH 6
#define MAX_VEHICLEDATA 6
#define MAX_GEARS_X 4
#define MAX_GEARS_Y 2
#define MAX_THROTTLE_X 5
#define MAX_THROTTLE_Y 2
#define MAX_BRAKE_X 4
#define MAX_BRAKE_Y 2

struct
{
  /* engine data */
  double emap[MAX_EMAP_Y][MAX_EMAP_X];
  double nevec[MAX_NEVEC];
  double thvec[MAX_THVEC];

  /* torque converter data */
  double converter_data[MAX_CONVERTER_DATA_Y][MAX_CONVERTER_DATA_X];

  /* transmission data */
  double gears[MAX_GEARS_Y][MAX_GEARS_X];
  double downtab[MAX_DOWNTAB_Y][MAX_DOWNTAB_X];
  double downth[MAX_DOWNTH];
  double uptab[MAX_UPTAB_Y][MAX_UPTAB_X];
  double upth[MAX_UPTH];
  double tw; /* shift time */

  /* driver inputs */
  double throttle[MAX_THROTTLE_Y][MAX_THROTTLE_X];

} pwr; /* values in ft,lbs,sec */

struct
{
  /* vehicle data */
  double vehicledata[MAX_VEHICLEDATA];
  double ie;   /* engine inertia */
  double rfd;   /* final drive ratio */
  double rload0;   /* rolling resistance */
  double rload2;   /* air drag coefficient */
  double rw;   /* wheel radius */
  double iv;   /* vehicle inertia */
```

```
  /* driver inputs */
  double brake[MAX_BRAKE_Y][MAX_BRAKE_X];

} drvl; /* values in ft,lbs,sec */

#define RK_ORDER 4

struct
{
  /* engine data */
  double p2;   /* engine RPM */
  double cet[RK_ORDER];   /* engine Torque */

  /* torque converter data */
  double sr[RK_ORDER];   /* speed ratio */
  double ck[RK_ORDER];   /* capacity factor */
  double crtq[RK_ORDER];   /* torque ratio */
  double Ti[RK_ORDER];   /* impeller torque */

  /* transmission data */
  double gearstate;   /* gear */
  double crtr;   /* transmission ratio */
  double tdn;   /* time delay for downshift */
  double tup;   /* time delay for upshift */
  int flgd;   /* flag indicator in downshift mode */
  int flgu;   /* flag indicator in upshift mode */

  /* driver inputs */
  double cthrottle[RK_ORDER];   /* throttle input */

  /* temp variables */
  double down_threshold;
  double up_threshold;

} pwr_state;

struct
{
  /* vehicle data */
  double p9;   /* wheel RPM */
  double Tload[RK_ORDER];   /* Load Torque */
  double vspd[RK_ORDER];   /* vehicle speed */

  /* driver inputs */
  double cbrake[RK_ORDER];   /* brake input */

} drvl_state;

void update_pwr_state(double t,int i,double rk);
void update_drvl_state(double t,int i,double rk);
void shift_logic(double t,double *up,double *dn);
void init_pwr();
void init_drvl();
void drvlmod(double t, double h);
void pwrmod(double t, double h);

/* ----PROGRAM START----*/
void main()
{
  /* ---VARIABLES
     fid1,fid2    file handles for misc. output
     h            time step increment
     t            time
     end          end time
     ierr         function return code
  */

  FILE *fid1,*fid2;
  double h,t,end;
  int ierr;

  init_drvl();  /* make sure to do this first--init_pwr uses drvl.ie for initial p2*/
  init_pwr();

  fid1 = fopen("out1.txt","wt");
  fid2 = fopen("out2.txt","wt");
  h = 0.01;  /* set time step */
```

```
  end = 200.0;  /* end after 200 seconds */

  /* ---MAIN LOOP--- */
  t = 0.0;
  for (t = 0.0; t <= 200.0; t = t + h)
  {
    pwrmod(t,h);
    drvlmod(t,h);
    shift_logic(t,&pwr_state.up_threshold,&pwr_state.down_threshold);

    fprintf(fid1,"%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf\n",
        t,pwr_state.p2/drvl.ie, drvl_state.vspd[0],pwr_state.cthrottle[0],drvl_state.cbrake[0],
        pwr_state.cet[0],pwr_state.crtr,pwr_state.sr[0],pwr_state.ck[0],pwr_state.crtq[0],pwr_state.Ti[0],
        drvl_state.Tload[0],pwr_state.gearstate,pwr_state.up_threshold,pwr_state.down_threshold);
    printf("gear,th,up,sp,dn=%lf %lf %lf %lf %lf\n",pwr_state.gearstate,pwr_state.cthrottle[0],
        pwr_state.up_threshold,drvl_state.vspd[0],pwr_state.down_threshold);
  }
  fclose(fid1);
  fclose(fid2);
};
```

```
void init_pwr()
{
  /* ---INITIALIZE THE PWR AND PWR_STATE STRUCTURE DATA - UNITS ARE FT,LBS,SEC,RPM--- */
  int dim[3];  /* dim[0] = # of dimensions, dim[1] = size of dim1, dim[2] = size of dim2 */
  int ierr;  /* error return code */

  dim[0] = 2;
  dim[1] = MAX_EMAP_Y;
  dim[2] = MAX_EMAP_X;
  ierr = readtbl2d("emap.arr",dim,pwr.emap);
  /* ierr = printtbl2d(dim,pwr.emap); */

  dim[0] = 1;
  dim[1] = MAX_NEVEC;
  dim[2] = 0;
  ierr = readtbl1d("nevec.arr",dim,pwr.nevec);
  /* ierr = printtbl1d(dim,pwr.nevec); */

  dim[0] = 1;
  dim[1] = MAX_THVEC;
  dim[2] = 0;
  ierr = readtbl1d("thvec.arr",dim,pwr.thvec);
  /* ierr = printtbl1d(dim,pwr.thvec); */

  dim[0] = 2;
  dim[1] = MAX_CONVERTER_DATA_Y;
  dim[2] = MAX_CONVERTER_DATA_X;
  ierr = readtbl2d("converter_data.arr",dim,pwr.converter_data);
  /* ierr = printtbl2d(dim,pwr.converter_data); */

  dim[0] = 2;
  dim[1] = MAX_GEARS_Y;
  dim[2] = MAX_GEARS_X;
  ierr = readtbl2d("gears.arr",dim,pwr.gears);
  /* ierr = printtbl2d(dim,pwr.gears); */

  dim[0] = 2;
  dim[1] = MAX_DOWNTAB_Y;
  dim[2] = MAX_DOWNTAB_X;
  ierr = readtbl2d("downtab.arr",dim,pwr.downtab);
  /* ierr = printtbl2d(dim,pwr.downtab); */

  dim[0] = 1;
  dim[1] = MAX_DOWNTH;
  dim[2] = 0;
  ierr = readtbl1d("downth.arr",dim,pwr.downth);
  /* ierr = printtbl1d(dim,pwr.downth); */

  dim[0] = 2;
  dim[1] = MAX_UPTAB_Y;
  dim[2] = MAX_UPTAB_X;
  ierr = readtbl2d("uptab.arr",dim,pwr.uptab);
  /* ierr = printtbl2d(dim,pwr.uptab); */

  dim[0] = 1;
  dim[1] = MAX_UPTH;
  dim[2] = 0;
  ierr = readtbl1d("upth.arr",dim,pwr.upth);
  /* ierr = printtbl1d(dim,pwr.upth); */

  dim[0] = 2;
  dim[1] = MAX_THROTTLE_Y;
  dim[2] = MAX_THROTTLE_X;
  ierr = readtbl2d("throttle.arr",dim,pwr.throttle);
  /* ierr = printtbl2d(dim,pwr.throttle); */

  pwr_state.gearstate = 1.0;  /* start in 1st gear */
  ierr = interp1(&pwr.gears[0][0],&pwr.gears[1][0],MAX_GEARS_X,pwr_state.gearstate,&pwr_state.crtr);  /* set
transmission ratio */
  pwr.tw = 0.08;  /* set the shift time delay */

  pwr_state.tdn = 0.0;  /* initialize down shift time to zero */
  pwr_state.tup = 0.0;  /* initialize up shift time to zero */
  pwr_state.flgd = 0;  /* initialize downshift state flag to zero */
  pwr_state.flgu = 0;  /* initialize upshift state flag to zero */

  pwr_state.p2 = 1000.0 * drvl.ie;  /* initial engine speed to 1000 RPM */
```

```
  //printf("pwr_state.pw=%lf\n",pwr_state.p2);

  ///* temp vars */
  //pwr_state.down_threshold=0.0;
  //pwr_state.up_threshold=0.0;
  /*pwr_state.aa[RK_ORDER]=0.0; */

}
```

```
void init_drvl()
{
  /* ---INITIALIZE THE DRVL AND DRVL_STATE STRUCTURE DATA - UNITS ARE FT,LBS,SEC,RPM--- */
  int dim[3];  /* dim[0] = # of dimensions, dim[1] = size of dim1, dim[2] = size of dim2 */
  int ierr; /* error return code */

  dim[0] = 1;
  dim[1] = MAX_VEHICLEDATA;
  dim[2] = 0;
  ierr = readtbl1d("vehicledata.arr",dim,drvl.vehicledata);
  /* ierr = printtbl1d(dim,drvl.vehicledata); */

  dim[0] = 2;
  dim[1] = MAX_BRAKE_Y;
  dim[2] = MAX_BRAKE_X;
  ierr = readtbl2d("brake.arr",dim,drvl.brake);
  /* ierr = printtbl2d(dim,drvl.brake); */

  drvl.ie = drvl.vehicledata[0];  /* engine speed RPM */
  drvl.rfd = drvl.vehicledata[1];  /* final drive ratio */
  drvl.rload0 = drvl.vehicledata[2];  /* road load - resistance */
  drvl.rload2 = drvl.vehicledata[3];  /* road load - aerodynamic drag */
  drvl.rw = drvl.vehicledata[4];  /* tire radius */
  drvl.iv = drvl.vehicledata[5];  /* vehicle inertia */

  drvl_state.p9 = 0.0;  /* initialize wheel speed to zero */

}
```

```
void update_pwr_state(double t,int i,double rk)
{
  /* ---UPDATE THE PWR_STATE STRUCTURE DATA--- */
  int ierr;  /* error return code */

  /* ---UPDATE THROTTLE, ENGINE TRQ, SPEED RATIO, K-FACTOR, TRQ RATIO, IMPELLER TRQ--- */
  ierr = interp1(&pwr.throttle[0][0],&pwr.throttle[1][0],MAX_THROTTLE_X,t,&pwr_state.cthrottle[i]);
  ierr = interp2(pwr.nevec,pwr.thvec,pwr.emap,MAX_NEVEC,MAX_THVEC,(pwr_state.p2+rk)/drvl.ie,
                 pwr_state.cthrottle[i],&pwr_state.cet[i]);
  pwr_state.sr[i] = (pwr_state.crtr*drvl_state.p9*drvl.rfd/drvl.iv)/((pwr_state.p2+rk)/drvl.ie) ;
  ierr = interp1(&pwr.converter_data[0][0],&pwr.converter_data[1][0],MAX_CONVERTER_DATA_X,pwr_state.sr[i],
                 &pwr_state.ck[i] );
  ierr = interp1(&pwr.converter_data[0][0],&pwr.converter_data[2][0],MAX_CONVERTER_DATA_X,pwr_state.sr[i],
                 &pwr_state.crtq[i] );
  pwr_state.Ti[i] = pow( ((pwr_state.p2+rk)/drvl.ie)/pwr_state.ck[i] ,2.0);

}
```

```
void update_drvl_state(double t,int i,double rk)
{
  /* ---UPDATE THE DRVL_STATE STRUCTURE DATA--- */
  double vs;  /* sign of vehicle speed */
  int ierr;  /* error return code */

  /* ---UPDATE BRAKE FORC, VEHICLE SPEED, LOAD TRQ--- */
  ierr = interp1(&drvl.brake[0][0],&drvl.brake[1][0],MAX_BRAKE_X,t,&drvl_state.cbrake[i]);
  drvl_state.vspd[i] = 2.0*PI*drvl.rw*(60.0/5280.0)*(drvl_state.p9+rk)/drvl.iv; /* RPM - > MPH */

  if (drvl_state.vspd[i] >= 0.0)
  {
    vs = 1.0;
  }
  else
  {
    vs = -1.0;
  }
  drvl_state.Tload[i] = vs*(drvl.rload0+drvl.rload2*pow(drvl_state.vspd[i],2.0)+drvl_state.cbrake[i]);

}
```

```
void shift_logic(double t,double *up,double *dn)
{
  /* ---DETERMINE SHIFT STATE--- */
  int ierr;  /* error return code */

  /* ---SET SHIFT UP AND SHIFT DOWN SPEEDS--- */
  ierr = interp2(&pwr.gears[0][0],pwr.downth,pwr.downtab,MAX_GEARS_X,MAX_DOWNTH,pwr_state.gearstate,
                 pwr_state.cthrottle[0],dn);
  ierr = interp2(&pwr.gears[0][0],pwr.upth,pwr.uptab,MAX_GEARS_X,MAX_UPTH,pwr_state.gearstate,
                 pwr_state.cthrottle[0],up);

  if ((pwr_state.flgd != 1) && (drvl_state.vspd[0]<*dn))
  {
    /* ---ENTER DOWN SHIFT MODE--- */
    pwr_state.tdn = t;  /* current time */
    pwr_state.flgd = 1;  /* flag */
  }
  if ((pwr_state.flgd == 1) && (drvl_state.vspd[0]>*dn)) pwr_state.flgd=0; /* exit down shift */
  if ((pwr_state.flgd == 1) && (t-pwr_state.tdn >= pwr.tw) & (drvl_state.vspd[0]<=*dn))  /* shift down after tw
seconds */
  {
    /* ---SHIFT DOWN--- */
    if (pwr_state.gearstate==4.0) pwr_state.gearstate = pwr_state.gearstate-1.0;
    else if (pwr_state.gearstate==3.0) pwr_state.gearstate = pwr_state.gearstate-1.0;
    else if (pwr_state.gearstate==2.0) pwr_state.gearstate = pwr_state.gearstate-1.0;
  }

  if ((pwr_state.flgu != 1) && (drvl_state.vspd[0]>*up))
  {
    /* ---ENTER UP SHIFT MODE--- */
    pwr_state.tup = t;  /* current time */
    pwr_state.flgu = 1;  /* flag */
  }
  if ((pwr_state.flgu == 1) && (drvl_state.vspd[0]<*up)) pwr_state.flgu = 0;  /* exit up shift */
  if ((pwr_state.flgu==1) && (t-pwr_state.tup >= pwr.tw) && (drvl_state.vspd[0]>=*up))  /* shift up after tw
seconds */
  {
    /* ---SHIFT UP--- */
    if (pwr_state.gearstate==3.0) pwr_state.gearstate = pwr_state.gearstate+1.0;
    else if (pwr_state.gearstate==2.0) pwr_state.gearstate = pwr_state.gearstate+1.0;
    else if (pwr_state.gearstate==1.0) pwr_state.gearstate = pwr_state.gearstate+1.0;
  }

}
```

```
void pwrmod(double t, double h)
{
  /* ---UPDATE PWR STATE VARIABLE P2- ENGINE RPM--- */
  double p2h;  /* temporary state variable */
  double aa[RK_ORDER];  /* Runge-Kutta(RK) Integration variable */
  int ierr;  /* error return code */

  ierr = interp1(&pwr.gears[0][0],&pwr.gears[1][0],MAX_GEARS_X,pwr_state.gearstate,&pwr_state.crtr);  /* update
the transmission ratio */

  /* ---UPDATE PWR STATE VARIABLE P2 FOR EACH STEP - 4TH ORDER RK--- */
  update_pwr_state(t,0,0.0);
  aa[0] = h*(pwr_state.cet[0]-pwr_state.Ti[0]);

  update_pwr_state(t+h/2.0,1,aa[0]/2.0);
  aa[1] = h*(pwr_state.cet[1]-pwr_state.Ti[1]);

  update_pwr_state(t+h/2.0,2,aa[1]/2.0);
  aa[2] = h*(pwr_state.cet[2]-pwr_state.Ti[2]);

  update_pwr_state(t+h,3,aa[2]);
  aa[3] = h*(pwr_state.cet[3]-pwr_state.Ti[3]);

  p2h = pwr_state.p2+(aa[0]+2.0*aa[1]+2.0*aa[2]+aa[3])/6.0;

  /* ---ENSURE PWR STATE VARIABLE P2 IS IN PROPER RANGE--- */
  if (p2h/drvl.ie > 6000.0)
  {
    p2h = 6000.0 * drvl.ie;  /* upper limit = 6000 RPM */
  }
  else if (p2h/drvl.ie < 600.0)
  {
    p2h = 600.0 * drvl.ie;  /* lower limit = 600 RPM */
  }

  /* ---UPDATE STATE--- */
  pwr_state.p2=p2h;

}
```

```
void drvlmod(double t,double h)
{
  /* ---UPDATE DRVL STATE VARIABLE P8-WHEEL RPM--- */
  double p9h;  /* temporary state variable */
  double aa[RK_ORDER];  /* Runge-Kutta(RK) Integration variable */
  int ierr;  /* error return code */

  /* ---UPDATE DRVL STATE VARIABLE P8 FOR EACH STEP - 4TH ORDER RK--- */
  /* if internal states not available or unknown use stage[0] to simulate only one or previous state */
  update_drvl_state(t,0,0.0);
  aa[0] = h*(drvl.rfd*pwr_state.crtr*pwr_state.crtq[0]*pwr_state.Ti[0] - drvl_state.Tload[0]);

  update_drvl_state(t+h/2.0,1,aa[0]/2.0);
  aa[1] = h*(drvl.rfd*pwr_state.crtr*pwr_state.crtq[1]*pwr_state.Ti[1] - drvl_state.Tload[1]);

  update_drvl_state(t+h/2.0,2,aa[1]/2.0);
  aa[2] = h*(drvl.rfd*pwr_state.crtr*pwr_state.crtq[2]*pwr_state.Ti[2] - drvl_state.Tload[2]);

  update_drvl_state(t+h,3,aa[2]);
  aa[3] = h*(drvl.rfd*pwr_state.crtr*pwr_state.crtq[3]*pwr_state.Ti[3] - drvl_state.Tload[3]);

  p9h = drvl_state.p9+(aa[0]+2.0*aa[1]+2.0*aa[2]+aa[3])/6.0;

  /* ---UPDATE STATE--- */
  drvl_state.p9 = p9h;

}
```

Support Functions

```
/* interp1.c
   This is a 1D interpolation subroutine.  Given x,y and xi return
   interplated value yo.  Return function value is index s or -s if
   out of range.  If out of range, yo is extrapolated.

   02-03-27 Updated syntax.
   02-03-11 Converted to C from F90.  Indexes in C are zero "[0]" based.
            Adjust s, e, pvt's accordingly.
*/

#include <stdio.h>

int interp1(double *x, double *y, int l, double xi, double *yo)
{
  /* ---VARIABLES--- */
  int s;  /* start index */
  int e;  /* end index */
  int pvt1;  /* pivot pt 1 */
  int pvt2;  /* pivot pt 2 for even */
  int i;  /* process counter */

  s = 0;  /* start at very beginning */
  e = l - 1;  /* start at very end */

  /* check for range error */
  if (xi < *(x))
  {
    /* if no extrapolation  use -> *yo = *(y); */
    /* extrapolate */
    s = 0;
    e = s + 1;
    *yo = *(y + s) + (xi - *(x + s)) * ( *(y + e) - *(y + s) ) / ( *(x + e) - *(x + s) );
    return -s;  /* (-) means out of range */
  }
  else if ( xi > *(x + l - 1))
  {
    /* if no extrapolation use -> *yo = *(y + l - 1); */
    /* extrapolate */
    e = l - 1;
    s = e - 1;
    *yo = *(y + s) + (xi - *(x + s)) * ( *(y + e) - *(y + s) ) / ( *(x + e) - *(x + s) );
    return -s; /* (-) means out of range */
  }

  /* ---PROCESS LOOP--- */
  for (i = 0; i < l; i++)
  {
    if ((e - s) % 2 == 0 )
    {
      /* ---EVEN--- */
      pvt1 = s + (e - s) / 2;
      if (xi == *(x + pvt1))
      {
        *yo = *(y + pvt1);
        return pvt1;
      }
      if (xi > *(x + pvt1))
      {
        s = pvt1;  /* top half */
      }
      else
      {
        e = pvt1;  /* bottom half */
      }
    }
    else
    {
      /* ---ODD--- */
      pvt1 = s + (e - s - 1) / 2;
      pvt2 = pvt1 + 1;
      if (xi > *(x + pvt2))
      {
        s = pvt2;  /* top half */
      }
```

```
      else if (xi < *(x + pvt1))
      {
        e = pvt1;  /* bottom half */
      }
      else
      {
        s = pvt1;  /* between these */
        e = pvt2;
      }
    }
    if ((e - s) <= 1)
    {
      /* ---FINAL ANSWER--- */
      *yo = *(y + s) + (xi - *(x + s)) * ( *(y + e) - *(y + s) ) / ( *(x + e) - *(x + s) );
      return s;
    }
  }
  return -1;
}
```

```
/* interp2.c
   This is a 2D interpolation subroutine.  Given x,y,z,xi and yi return
   interplated value zo.  Return function value is -s if
   out of range. Uses interp1

   02-03-27 Updated syntax.
   02-03-11 Converted to C from F90.  Indexes in C are zero "[0]" based.
*/

#include <stdio.h>

/*int interp1(double *x, double *y, int l, double xi, double *yo)*/

int interp2(double *x, double *y, double *z, int m, int n, double xi, double yi, double *zo)
{
  /* ---VARIABLES--- */
  int sx;  /* start x index */
  int sy;  /* start y index */
//  int i;  /*
  int ierr;  /* error return code */
  double valy;  /* interpolated value of y1 */
  double valy1;  /* interpolated value of y2 */
  double tmp;  /* temporary variable */

  /* ---GET INDEXES--- */
  sx = interp1(x,x,m,xi,&tmp);
  sy = interp1(y,y,n,yi,&tmp);

  /* ---IF YOU DO NOT WANT EXTRAPOLATION UNCOMMENT THIS CODE---
   % if (sx==-1 || sy==-1)
   % {
   %   *(zo) = -1;
   %   return -1;
   %  }
   */

  /* ---GET Y'S FROM X'S--- */
  valy = *(z + sy*m + sx) + (xi - *(x + sx)) * ( *(z + sy*m + (sx+1)) - *(z + sy*m + sx) ) / ( *(x + (sx+1)) -
*(x + sx) );
  valy1 = *(z + (sy+1)*m + sx) + (xi - *(x + sx)) * ( *(z + (sy+1)*m + (sx+1)) - *(z + (sy+1)*m + sx) ) / ( *(x
+ (sx+1)) - *(x + sx) );

  /* ---GET Z FROM Y'S--- */
  *zo  = valy + (yi - *(y + sy)) * (valy1 - valy) / ( *(y + (sy+1)) - *(y + sy) );
  return 0;
}
```

```
/* readtbl1d.c
   Reads in table from file according to format.

   02-03-27 Update syntax.
   02-03-18 Created.

   ---INPUT FILE FORMAT---
   %%NDIM:# dimenstions
   %%DIMS:dim1 dim2 dim3 dim4 ...
   val()
   %comment
   val()
   ...

*/

#include <string.h> /*strncmp*/
#include <stdio.h>

int readtbl1d(const char *file, int *dim, double arr1d[])
{
  /* ---VARIABLES--- */
  FILE *fin;  /* file handle */
  int val;  /* scanned value */
  int ndim;  /* number of dimensions */
  int dims[2];  /* size of dimensions */
  char buff[1024];  /* read buffer */
  int row;  /* row counter */

  fin = fopen(file,"r");
  if (fin == NULL)
  {
    fclose(fin);
    return -1;
  }

  /* ---PROCESS HEADER--- */
  ndim = dims[0] = dims[1] = row = 0;
  while (fgets(buff,1024,fin) != NULL)
  {
    if (strncmp(buff,"%%",2)==0)
    {
      if (strncmp(&buff[2],"NDIM:",5)==0)
      {
        val = sscanf(&buff[7],"%d\n",&ndim);
      }
      else if (strncmp(&buff[2],"DIMS:",5)==0)
      {
        if (ndim == 1)
        {
          val = sscanf(&buff[7],"%d\n",&dims[0]);
        }
      }
    }
    else if(strncmp(buff,"%",1)!=0)
    {
      break;
    }
  }
  val = sscanf(&buff[0],"%lf\n",&arr1d[row]);
  row++;
  if ((ndim != dim[0]) || (dims[0] != dim[1])) return -1;

  /* ---PROCESS DATA--- */
  while (fgets(buff,1024,fin) != NULL)
  {
    if (strncmp(buff,"%",1)==0) continue;
    val = sscanf(&buff[0],"%lf\n",&arr1d[row]);
    row++;
    if (row >= dim[1]) break;
    /* check row bound here */
  }
  fclose(fin);
  return 0;
}
```

```
/* readtbl2d.c
   Reads in 2d table from file according to format.

   02-03-27 Update syntax.
   02-03-18 Created.

   ---INPUT FILE FORMAT---
   %%NDIM:# dimenstions
   %%DIMS:dim1 dim2 dim3 dim4 ...
   val()
   %comment
   val()
   ...

*/

#include <string.h> /*strncmp*/
#include <stdio.h>

int readtbl2d(const char *file, int *dim, double arr2d[][dim[2]])
{
  /* ---VARIABLES--- */
  FILE *fin;  /* file handle */
  int val;  /* scanned value */
  int ndim;  /* number of dimensions */
  int dims[2];  /* size of dimensions read in */
  char buff[1024];  /* read buffer */
  int row;  /* row counter */
  int col;  /* col counter */
  double dtmp;  /* temporary double */

  fin = fopen(file,"r");
  if (fin == NULL)
  {
    fclose(fin);
    return -1;
  }

  /* ---PROCESS HEADER--- */
  ndim = dims[0] = dims[1] = row = col = 0;
  while (fgets(buff,1024,fin) != NULL)
  {
    if (strncmp(buff,"%%",2)==0)
    {
      if (strncmp(&buff[2],"NDIM:",5)==0)
      {
        val = sscanf(&buff[7],"%d\n",&ndim);
      }
      else if (strncmp(&buff[2],"DIMS:",5)==0)
      {
        if (ndim == 2)
        {
          val = sscanf(&buff[7],"%d %d\n",&dims[0],&dims[1]);
        }
      }
    }
    else if(strncmp(buff,"%",1)!=0)
    {
      break;
    }
  }
  val = sscanf(&buff[0],"%lf\n",&dtmp);
  arr2d[row][col]=dtmp;
  col++;
  /* check col bound here */
  if ((ndim != dim[0]) || (dims[0] != dim[1]) || (dims[1] != dim[2]))
  {
    exit(0);
    return -1;
  }

  /* ---PROCESS DATA--- */
  while (fgets(buff,1024,fin) != NULL)
  {
    if (strncmp(buff,"%",1)==0) continue;
    val = sscanf(&buff[0],"%lf\n",&arr2d[row][col]);
    col++;
```

```
    if (col >= dim[2])
    {
      row++;
      col = 0;
      /* check row bound here */
    }
    if (row >= dim[1]) break;
  }
  fclose(fin);
  return 0;
}
```

```
/* printtbl.c
   Prints a table.

   02-03-27 Update syntax.
   02-03-18 Created.

*/

#include <stdio.h>

int printtbl1d(int *dim, double arr1d[])
{
  /* ---VARIABLES--- */
  int ndim;  /* number of dimensions */
  int dim1;  /* size of dim1 */
  int dim2;  /* size of dim2 - not used in 1d */
  int i;  /* loop counter */
  double tmp;  /* output value */

  ndim = *(dim);
  dim1 = *(dim+1);
  for (i = 0; i < dim1; i++)
  {
    tmp = arr1d[i];
    printf("%lf\n",tmp);
  }
  return 0;
}
```

```
/* printtbl2d.c
   Prints a 2d table.

   02-03-27 Update syntax.
   02-03-18 Created.

*/

#include <stdio.h>

int printtbl2d(int *dim, double arr2d[][dim[2]])
{
  /* ---VARIABLES--- */
  int ndim;  /* number of dimensions */
  int dim1;  /* size of dim1 */
  int dim2;  /* size of dim2 */
  int i,j;  /* loop counters */
  double tmp;  /* output value */

  ndim = *(dim);
  if ((ndim < 1) || (ndim > 2)) return -1;
  dim1 = *(dim+1);
  if (ndim == 2) dim2 = *(dim+2);
  for (i = 0; i < dim1; i++)
  {
    for (j = 0; j < dim2; j++)
    {
      tmp = arr2d[i][j];
      printf("%lf ",tmp);
    }
    printf("\n");
  }
  return 0;
}
```

APPENDIX E - Result Comparison

`load case1 simout` **or** `load case2 simout`

Setup Script

```
%28-MAR-02
addpath /r/tac3/rtswwb/matlab/func
load ./my5/my5 simout
x=read('out1.txt',15,0);
t=x(:,1);
rpm2=x(:,2);
rpm21=simout.signals.values(:,3);
vs=x(:,3);
vs1=simout.signals.values(:,7);
th=x(:,4);
th1=simout.signals.values(:,1);
bk=x(:,5);
bk1=simout.signals.values(:,2);
et=x(:,6);
et1=simout.signals.values(:,10);
sr=x(:,8);
sr1=simout.signals.values(:,11);
k=x(:,9);
k1=simout.signals.values(:,12);
tqr=x(:,10);
tqr1=simout.signals.values(:,13);
ti=x(:,11);
ti1=simout.signals.values(:,4);
tl=x(:,12);
g=x(:,13);
g1=simout.signals.values(:,5);
up=x(:,14);
up1=simout.signals.values(:,8);
dn=x(:,15);
dn1=simout.signals.values(:,9);
%
't,rpm2,vs,th,bk,et,sr,k,tqr,ti,tl,g,up,dn'
%
```

Output Figures Script

```
clf
h=plot(t,rpm2,'Color','r','LineStyle','-.');
hold
plot(t,rpm21,'Color','k','LineStyle','-');
legend('C','MATLAB')
xlabel('TIME (S)')
ylabel('ENGINE SPEED (RPM)');
%print('-djpeg99',[char(gg(i)) '.jpg'])
print('-djpeg99','rpm.jpg')

clf
h=plot(t,rpm2-rpm21,'Color','r','LineStyle','-.');
legend('C - MATLAB')
xlabel('TIME (S)')
ylabel('ERROR DIFFERENCE (RPM)');
%print('-djpeg99',[char(gg(i)) '.jpg'])
print('-djpeg99','rpme.jpg')


clf
h=plot(t,vs,'Color','r','LineStyle','-.');
hold
plot(t,vs1,'Color','k','LineStyle','-');
legend('C','MATLAB')
xlabel('TIME (S)')
ylabel('VEHICLE SPEED (MPH)');
print('-djpeg99','vs.jpg')

clf
h=plot(t,vs-vs1,'Color','r','LineStyle','-.');
legend('C - MATLAB')
xlabel('TIME (S)')
ylabel('ERROR DIFFERENCE (MPH)');
print('-djpeg99','vse.jpg')
```

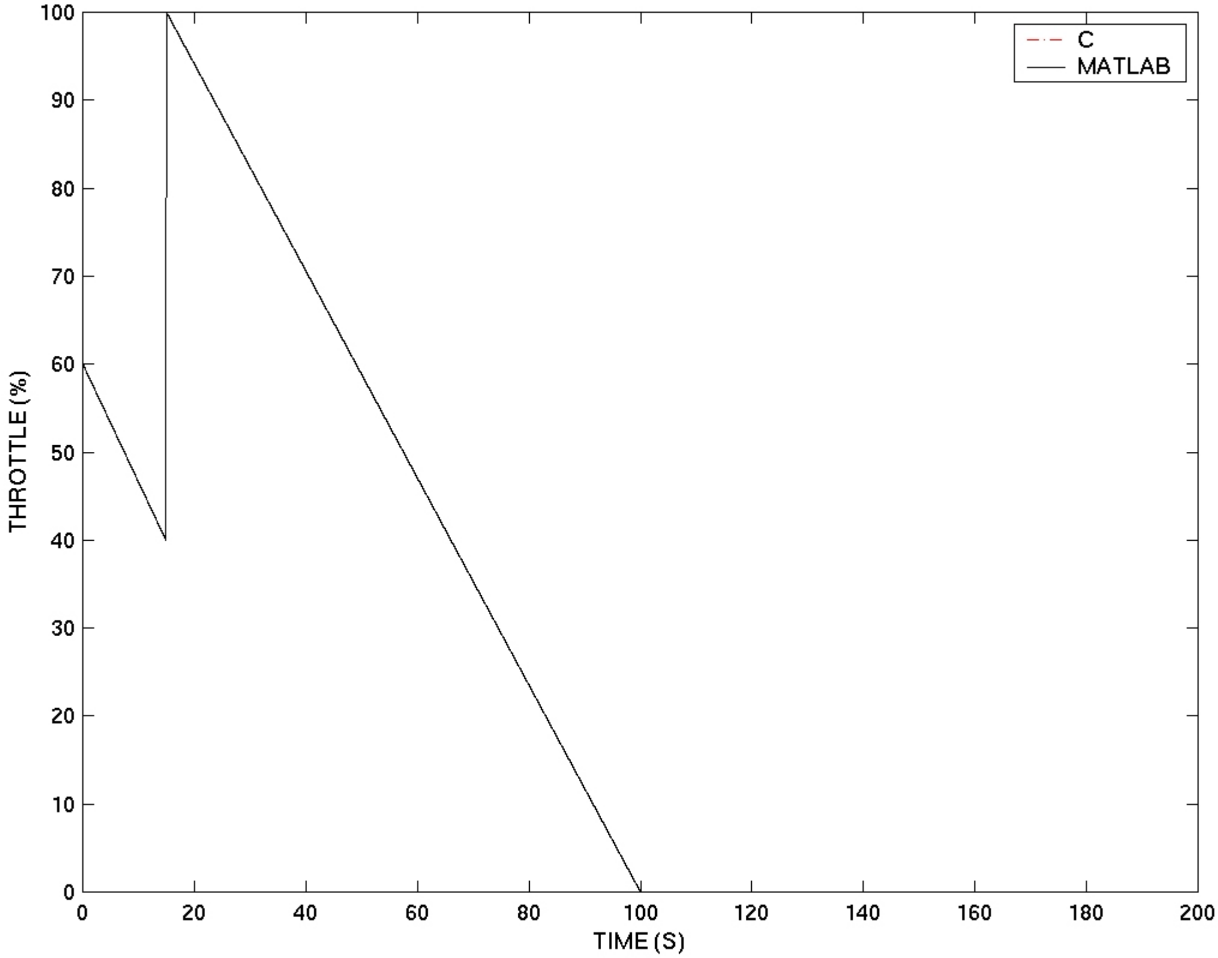
```
clf
h=plot(t,th,'Color','r','LineStyle','-.');
hold
plot(t,th1,'Color','k','LineStyle','-');
legend('C','MATLAB')
xlabel('TIME (S)')
ylabel('THROTTLE (%)');
print('-djpeg99','th.jpg')


clf
h=plot(t,bk,'Color','r','LineStyle','-.');
hold
plot(t,bk1,'Color','k','LineStyle','-');
legend('C','MATLAB')
xlabel('TIME (S)')
ylabel('BRAKE FORCE (LBF)');
print('-djpeg99','bk.jpg')



clf
h=plot(t,et,'Color','r','LineStyle','-.');
hold
plot(t,et1,'Color','k','LineStyle','-');
legend('C','MATLAB')
xlabel('TIME (S)')
ylabel('ENGINE TORQUE (LBF)');
print('-djpeg99','et.jpg')


clf
h=plot(t,sr,'Color','r','LineStyle','-.');
hold
plot(t,sr1,'Color','k','LineStyle','-');
legend('C','MATLAB')
xlabel('TIME (S)')
ylabel('SPEED RATIO');
print('-djpeg99','sr.jpg')


clf
h=plot(t,k,'Color','r','LineStyle','-.');
hold
plot(t,k1,'Color','k','LineStyle','-');
legend('C','MATLAB')
xlabel('TIME (S)')
ylabel('K FACTOR (RPM/sqrt(FT-LBF))');
print('-djpeg99','k.jpg')


clf
h=plot(t,tqr,'Color','r','LineStyle','-.');
hold
plot(t,tqr1,'Color','k','LineStyle','-');
legend('C','MATLAB')
xlabel('TIME (S)')
ylabel('TORQUE RATIO');
print('-djpeg99','trq.jpg')


clf
h=plot(t,ti,'Color','r','LineStyle','-.');
hold
plot(t,ti1,'Color','k','LineStyle','-');
legend('C','MATLAB')
xlabel('TIME (S)')
ylabel('IMPELLER TORQUE (LBF)');
print('-djpeg99','ti.jpg')


clf
h=plot(t,g,'Color','r','LineStyle','-.');
hold
plot(t,g1,'Color','k','LineStyle','-');
legend('C','MATLAB')
xlabel('TIME (S)')
```

```
ylabel('GEAR');
print('-djpeg99','g.jpg')


clf
h=plot(t,up,'Color','r','LineStyle','-.');
hold
plot(t,up1,'Color','k','LineStyle','-');
legend('C','MATLAB')
xlabel('TIME (S)')
ylabel('UP SHIFT THRESHOLD (MPH)');
print('-djpeg99','up.jpg')


clf
h=plot(t,dn,'Color','r','LineStyle','-.');
hold
plot(t,dn1,'Color','k','LineStyle','-');
legend('C','MATLAB')
xlabel('TIME (S)')
ylabel('DOWN SHIFT THRESHOLD (MPH)');
print('-djpeg99','dn.jpg')
```

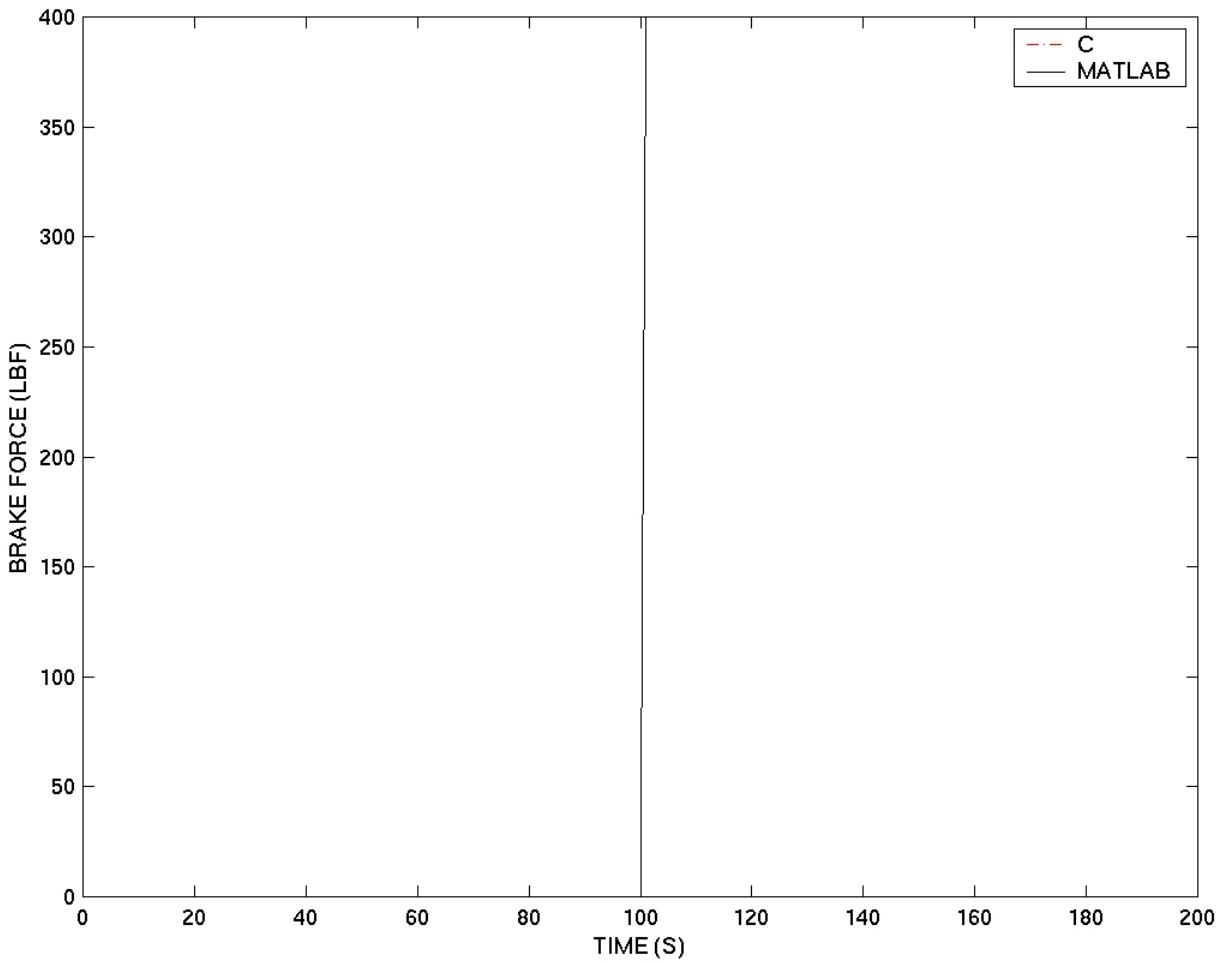
Read Function

```
%function [A]=read(file,col,skip) reads text file of values
% INPUT:
%  file - file name
%  col - number of data columns in file
%  skip - number of header lines to skip
%
% OUTPUT:
%  A - array of values
function [A]=READ(file,col,skip);
fid=fopen(file,'rt');
for i=1:skip,
  fgetl(fid);
end
A=fscanf(fid,'%lg',[col,inf])';
fclose(fid);
~
```

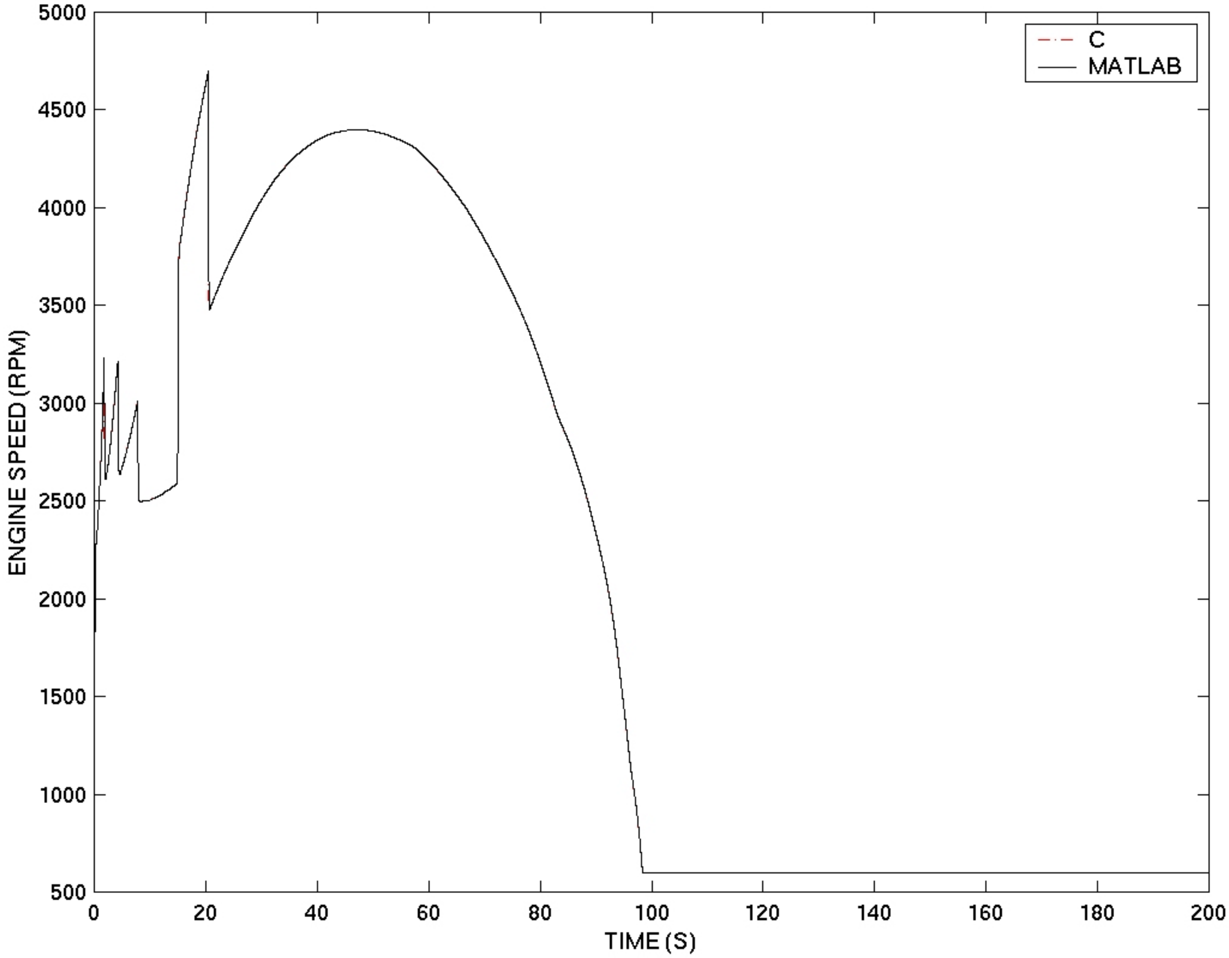
Case 1



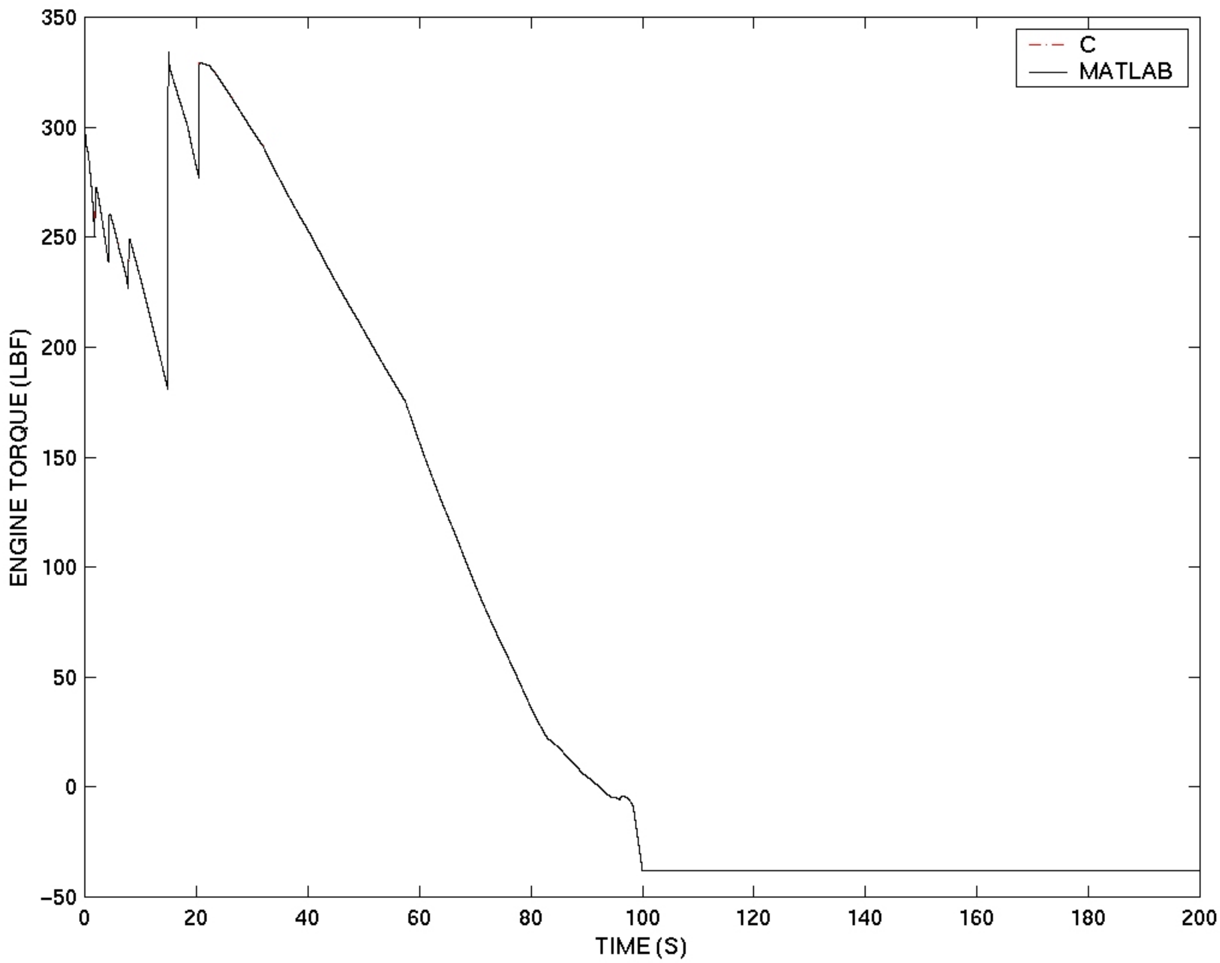
C
MATLAB
THROTTLE (%)
TIME (S)
0
10
20
30
40
50
60
70
80
90
100
0
20
40
60
80
100
120
140
160
180
200



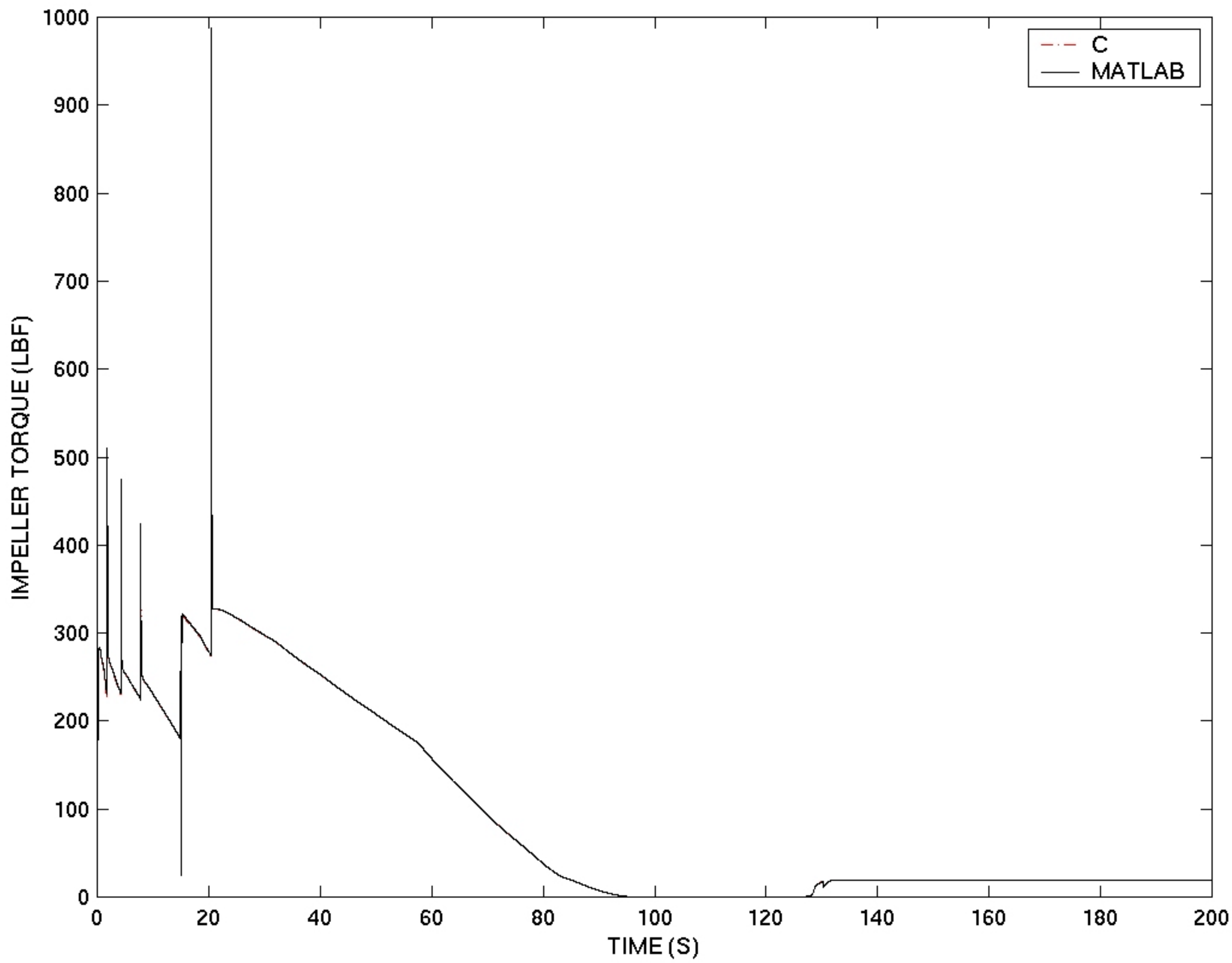
C
MATLAB
BRAKE FORCE (LBF)
TIME (S)
0
50
100
150
200
250
300
350
400
0
20
40
60
80
100
120
140
160
180
200



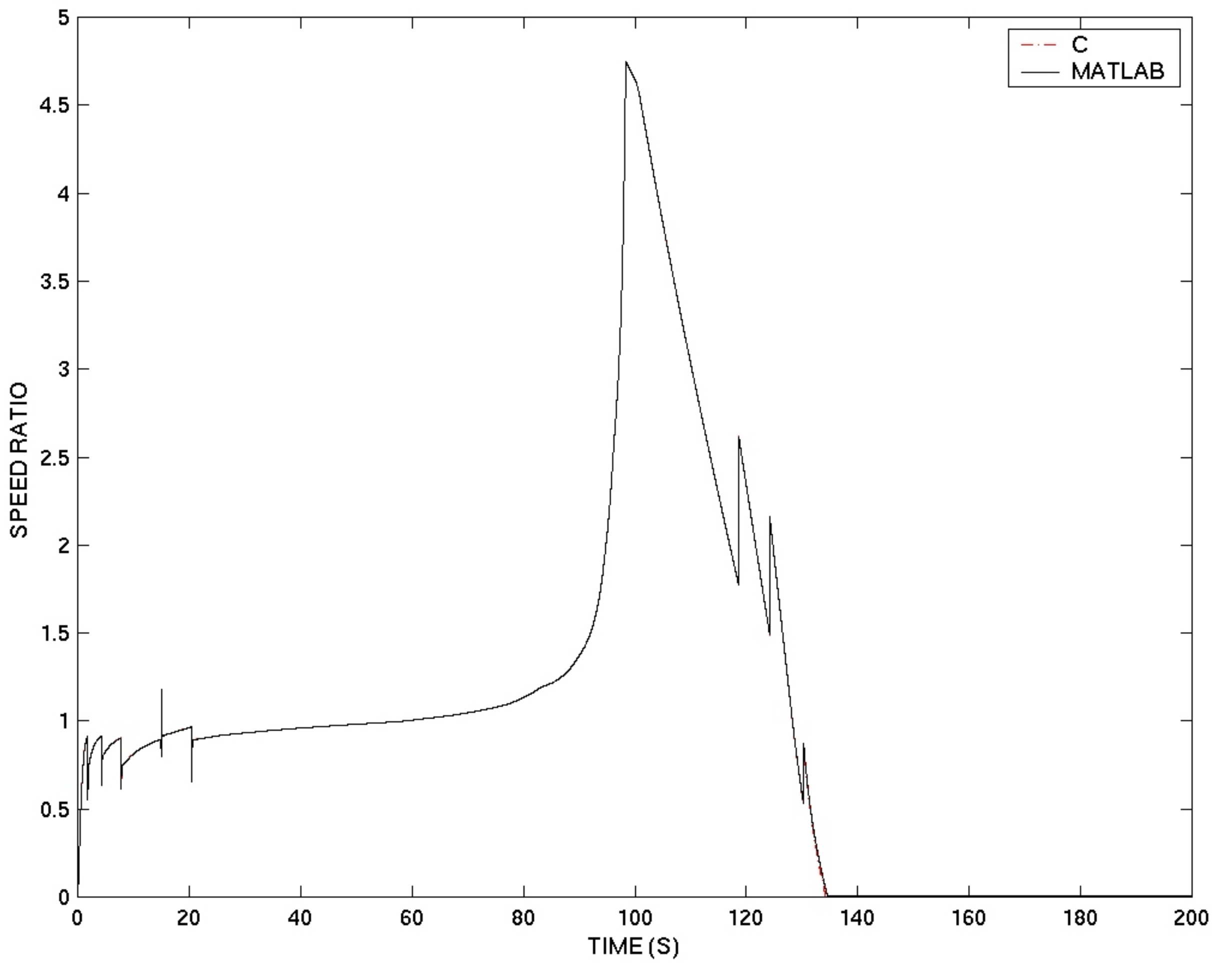
C
MATLAB
ENGINE SPEED (RPM)
TIME (S)
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
20
40
60
80
100
120
140
160
180
200



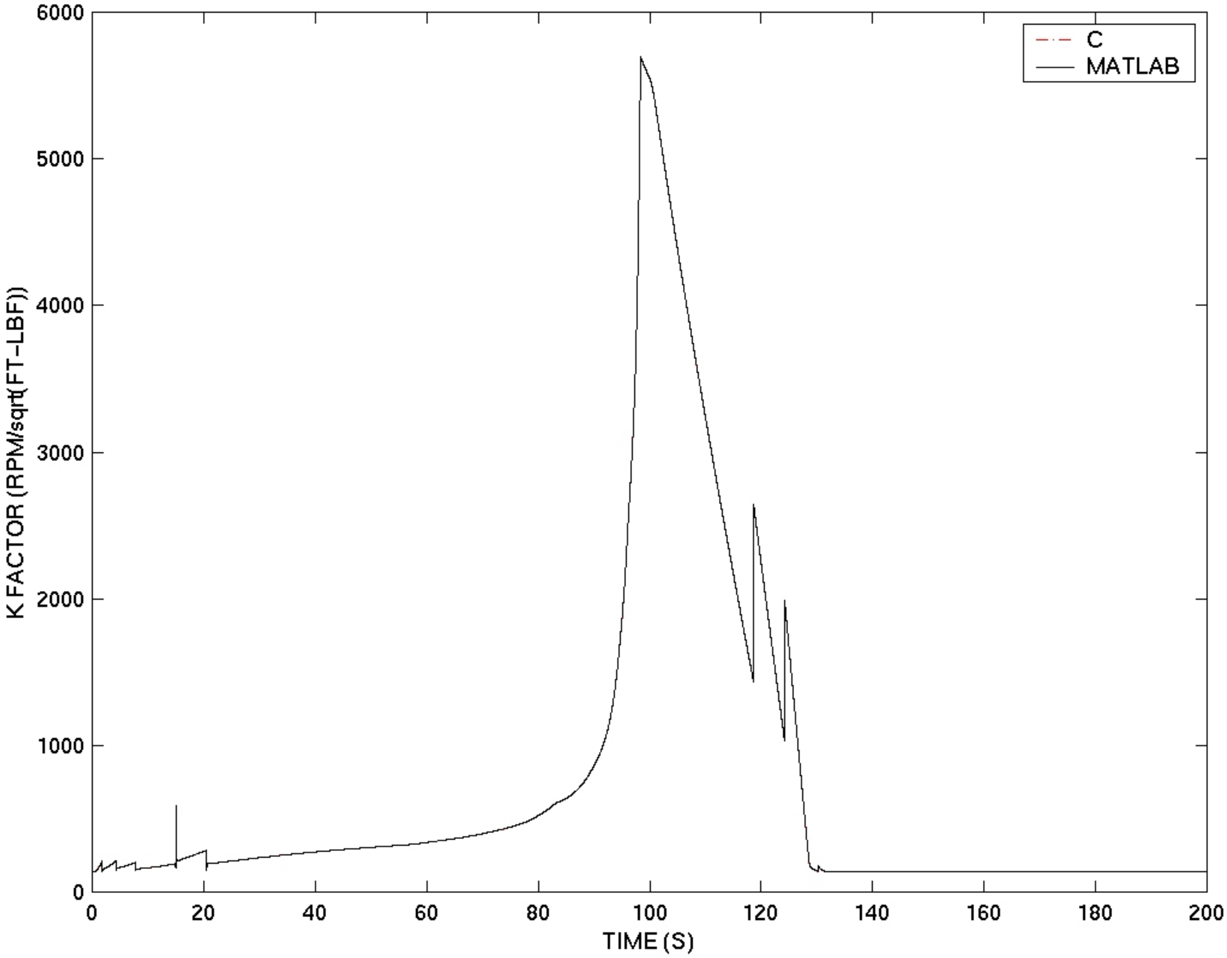
ENGINE TORQUE (LBF)
TIME (S)
C
MATLAB
350
300
250
200
150
100
50
0
−50
0
20
40
60
80
100
120
140
160
180
200



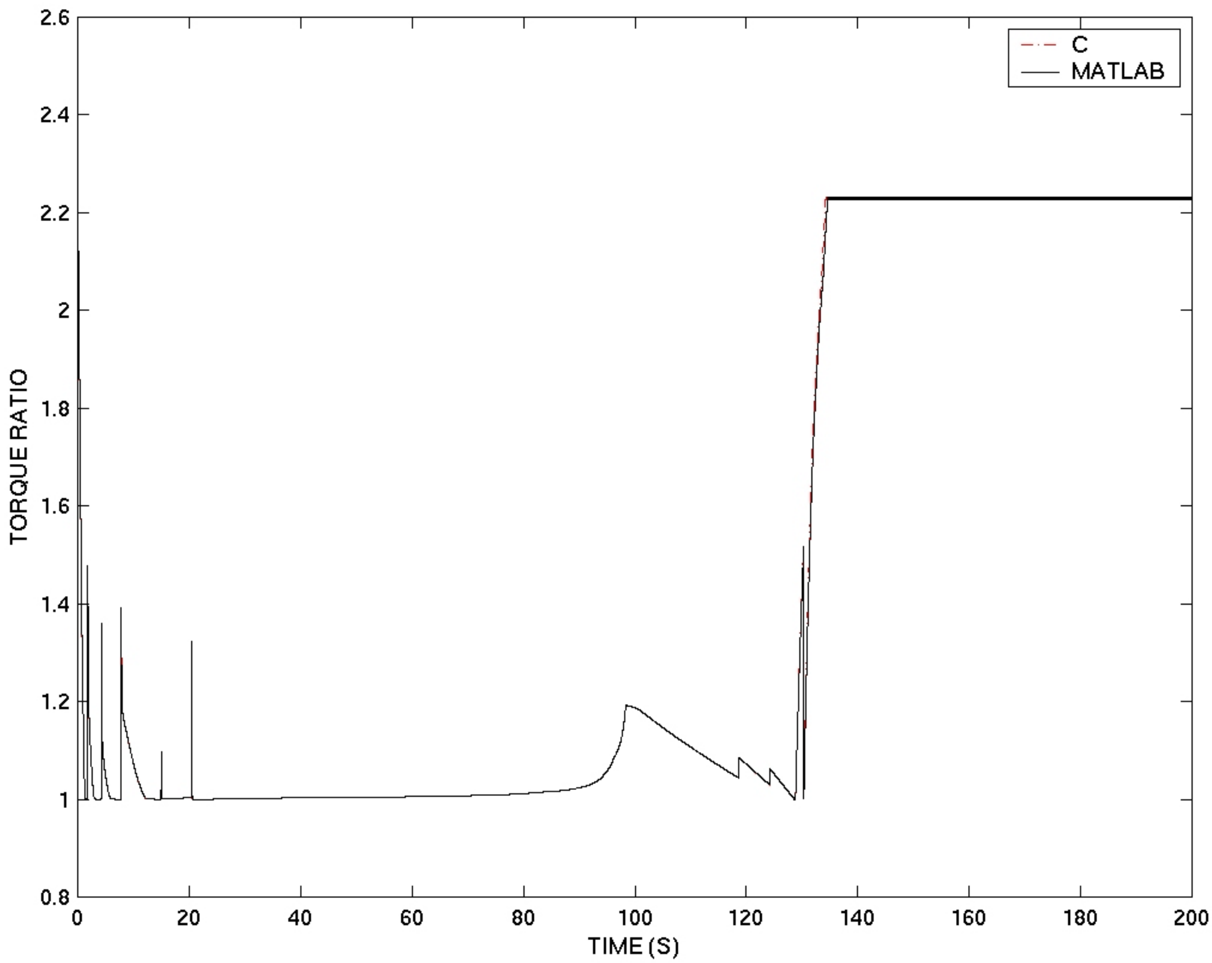
C
MATLAB
IMPELLER TORQUE (LBF)
TIME (S)
0
100
200
300
400
500
600
700
800
900
1000
0
20
40
60
80
100
120
140
160
180
200



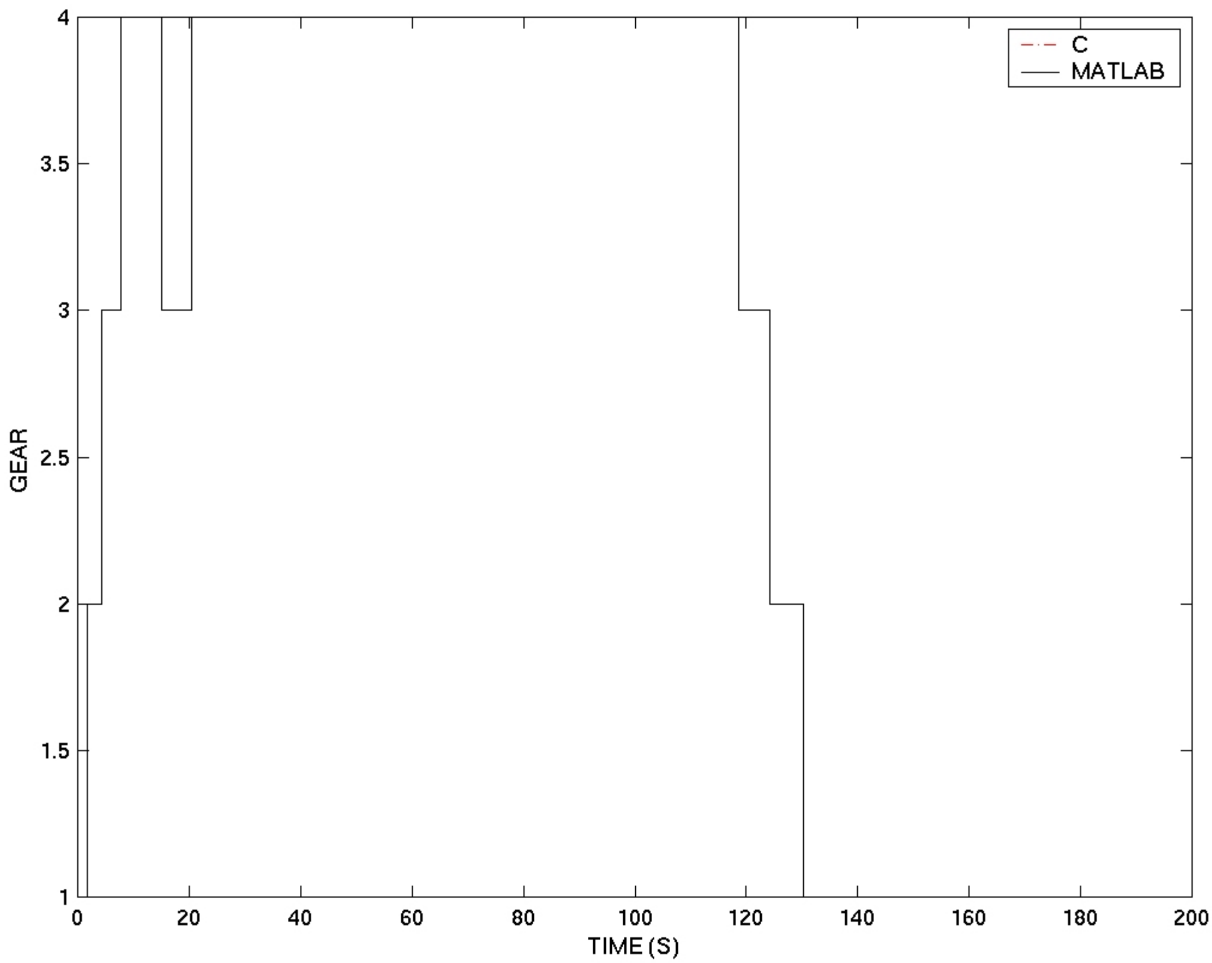
C
MATLAB
SPEED RATIO
TIME (S)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
20
40
60
80
100
120
140
160
180
200



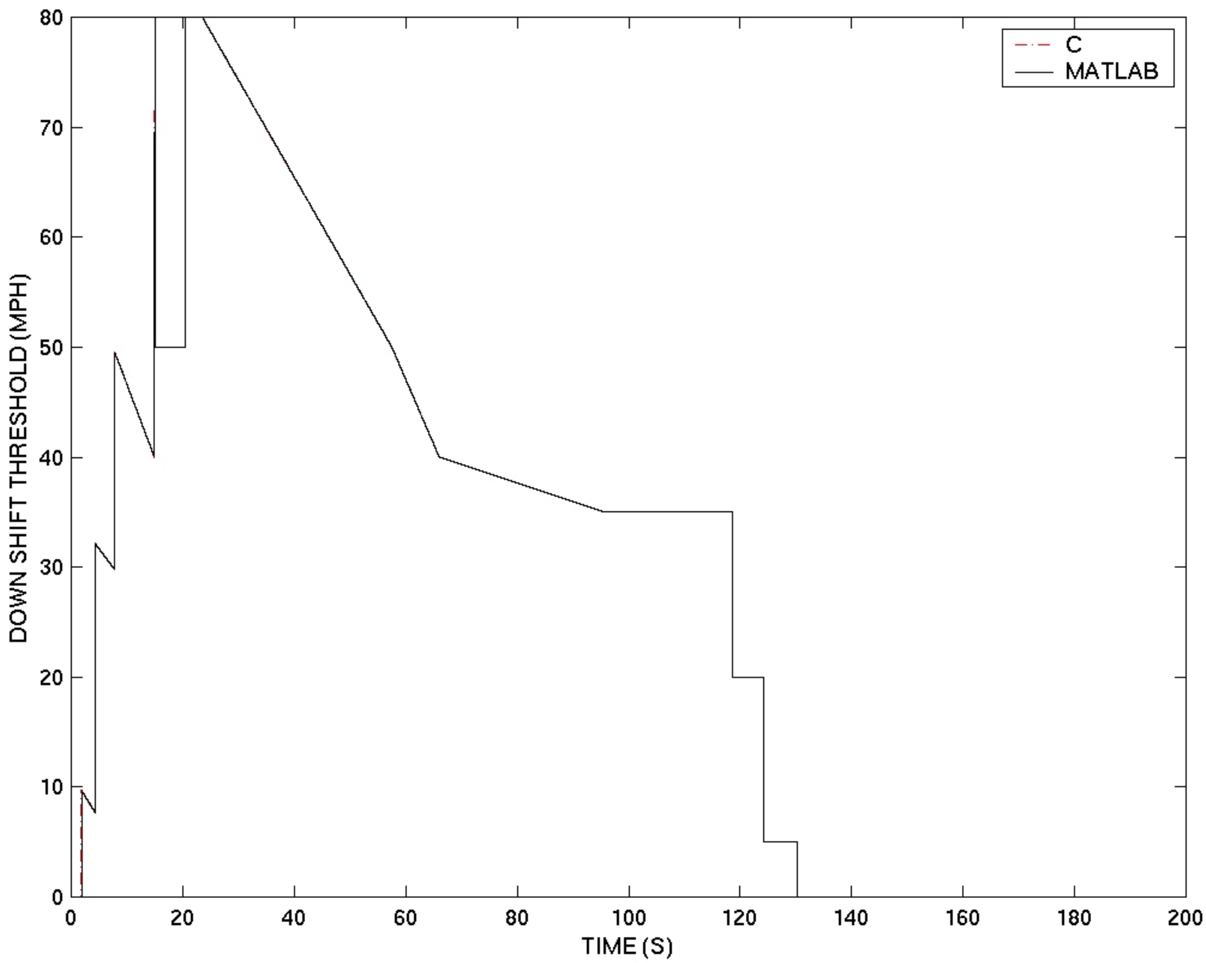
C
MATLAB
K FACTOR (RPM/sqrt(FT-LBF))
TIME (S)
0
1000
2000
3000
4000
5000
6000
0
20
40
60
80
100
120
140
160
180
200



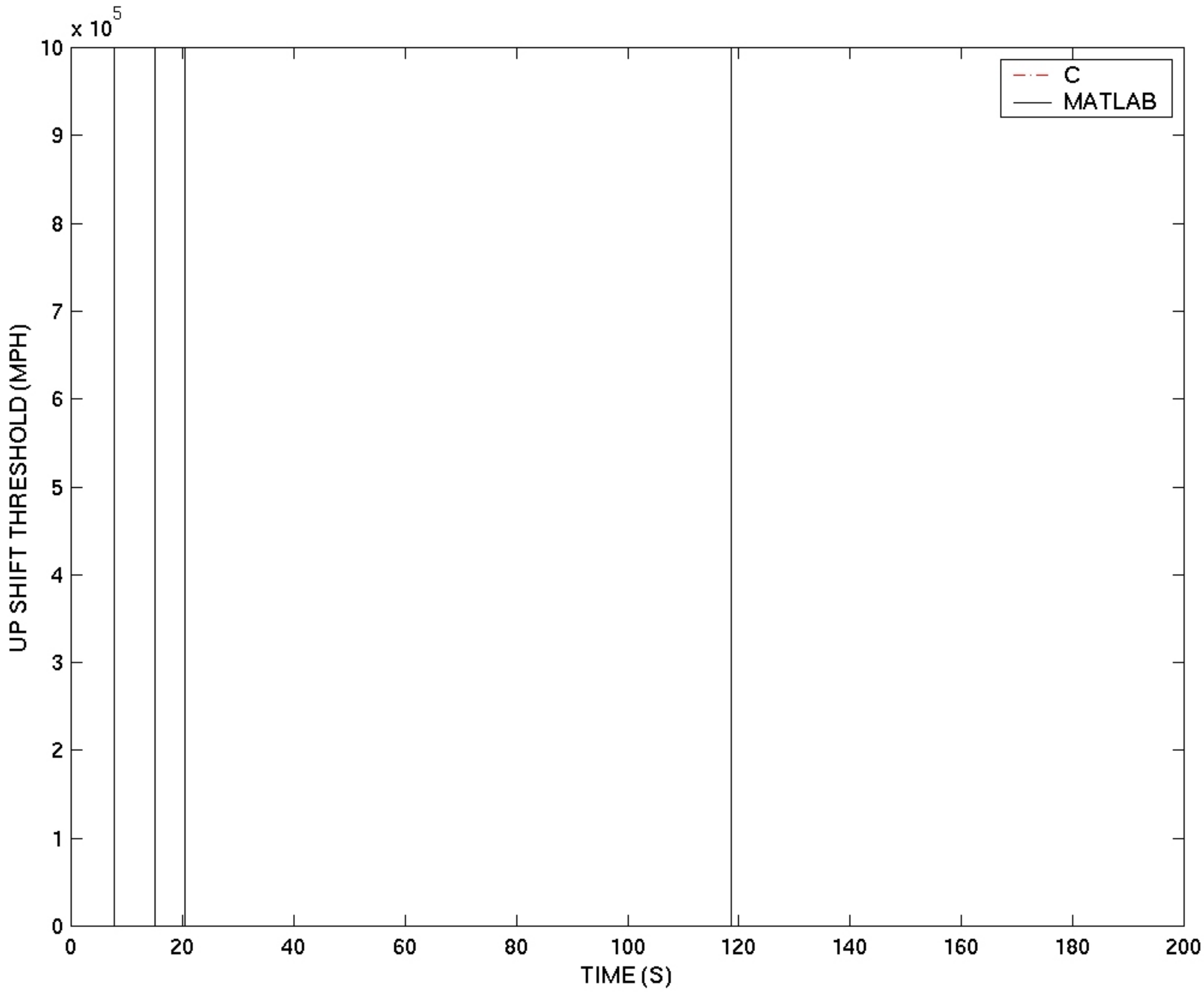
C
MATLAB
TORQUE RATIO
TIME (S)
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0
20
40
60
80
100
120
140
160
180
200

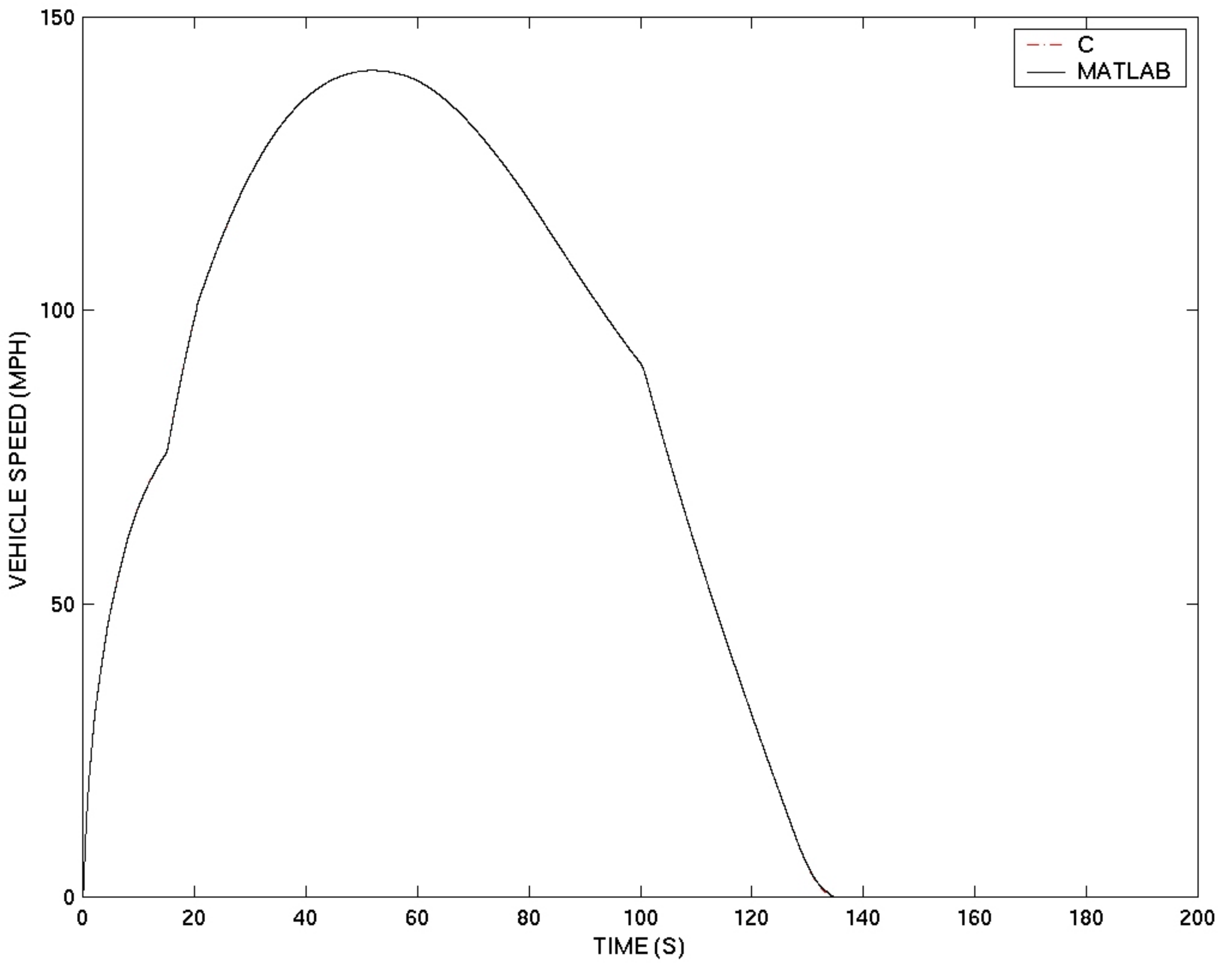


C
MATLAB
GEAR
TIME (S)

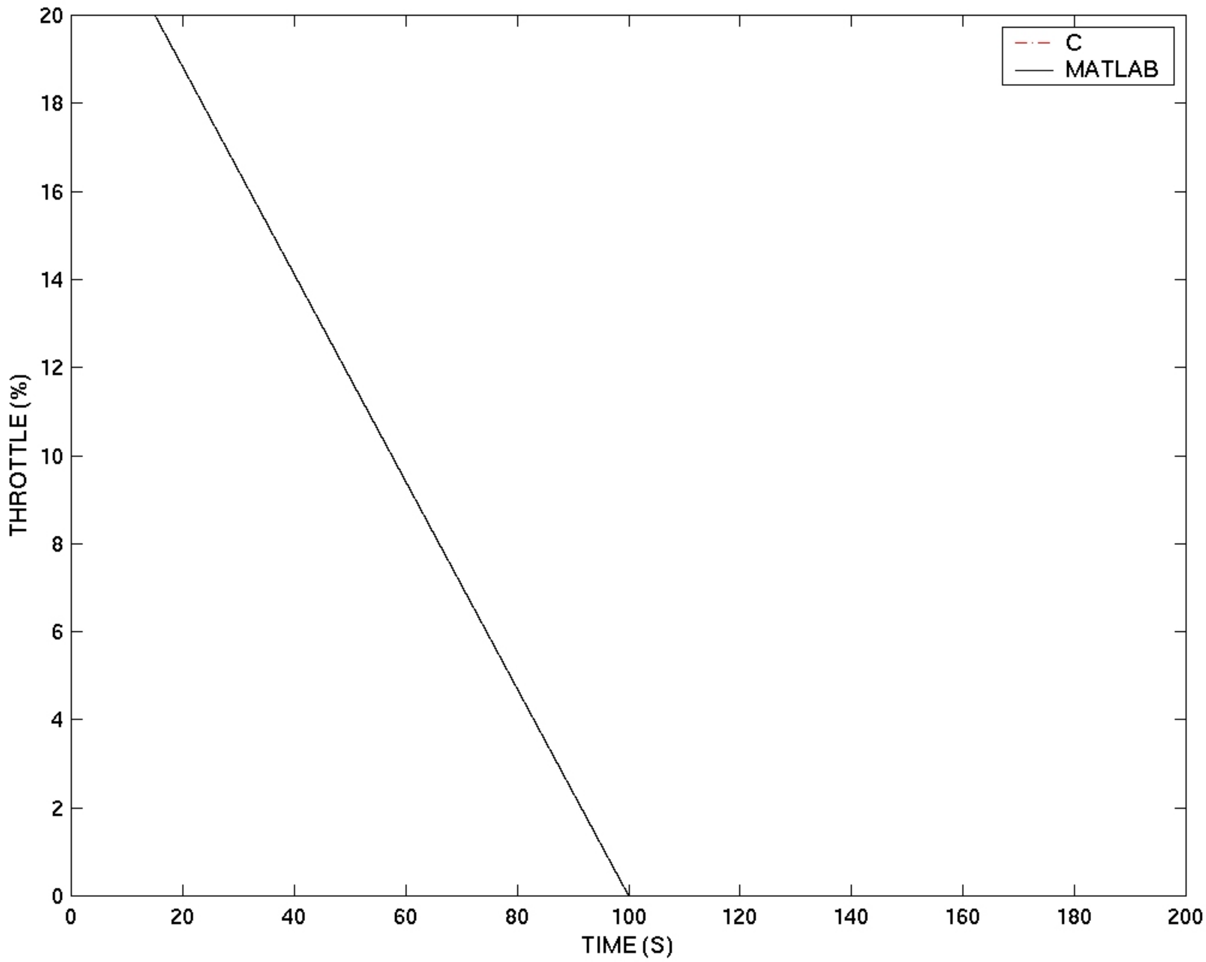


80
70
60
50
40
30
20
10
0
0
20
40
60
80
100
120
140
160
180
200
DOWN SHIFT THRESHOLD (MPH)
TIME (S)
C
MATLAB

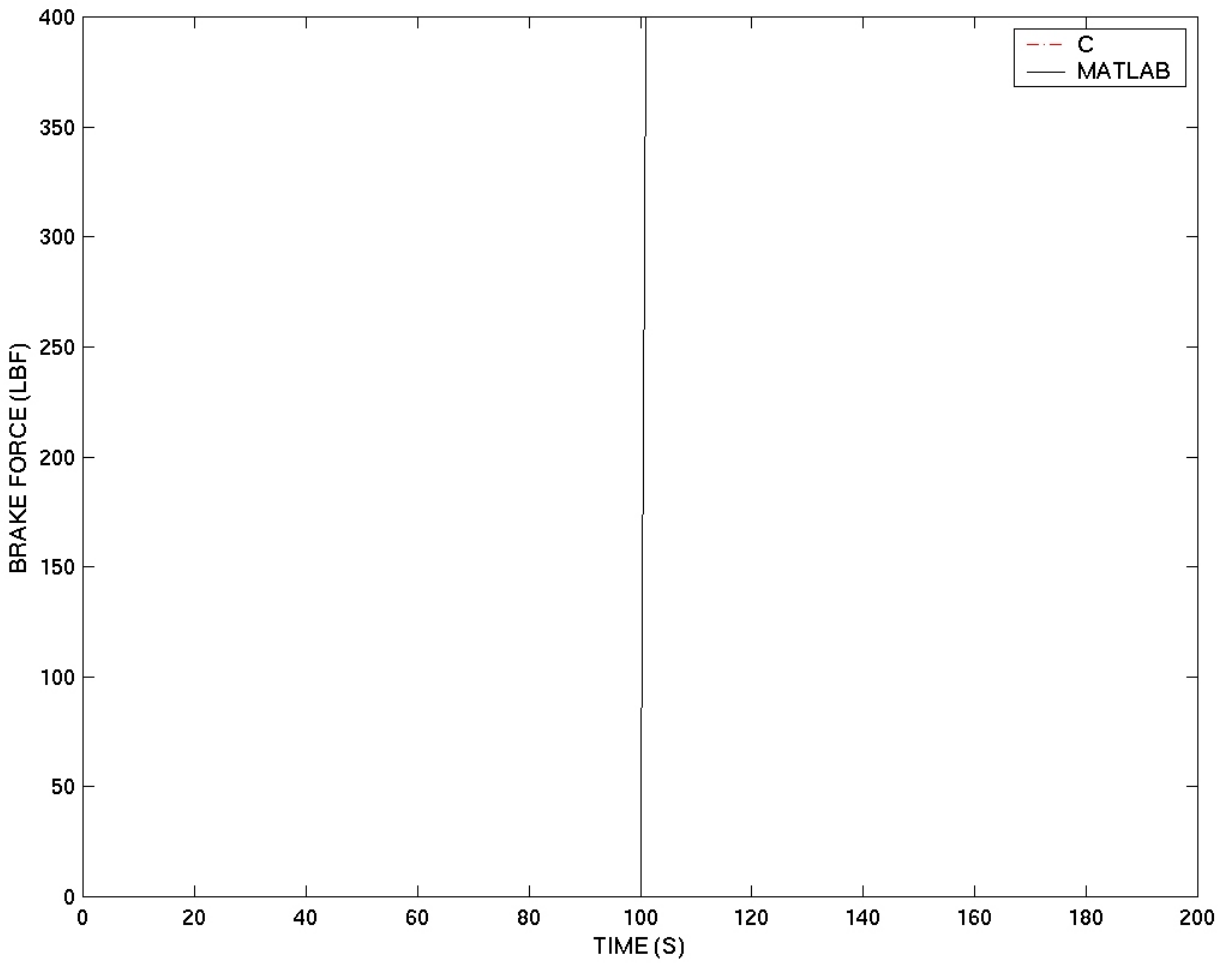

x 10^5
10
9
8
7
6
5
4
3
2
1
0
UP SHIFT THRESHOLD (MPH)
0
20
40
60
80
100
120
140
160
180
200
TIME (S)
C
MATLAB

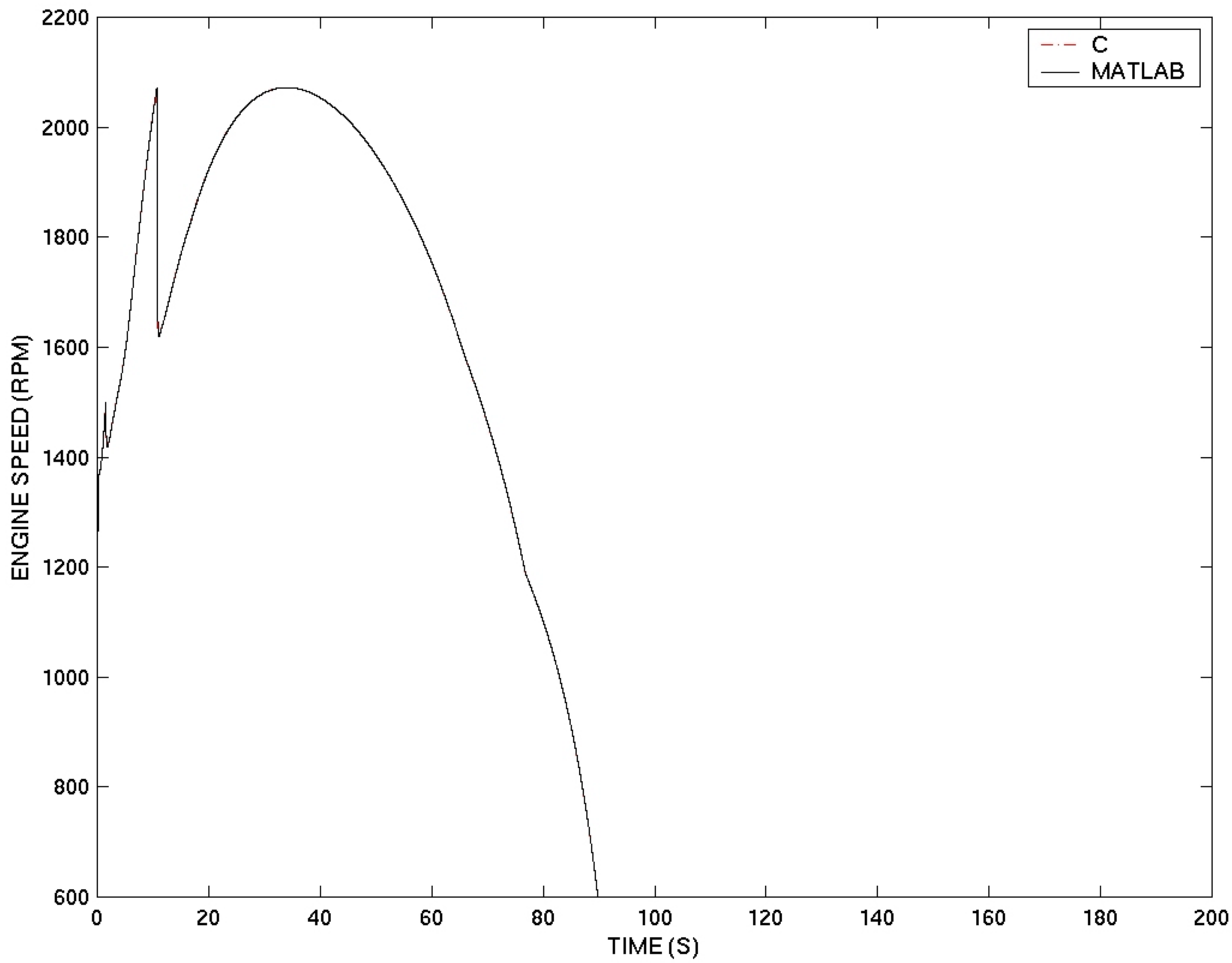

C
MATLAB
150
100
50
0
VEHICLE SPEED (MPH)
0
20
40
60
80
100
120
140
160
180
200
TIME (S)

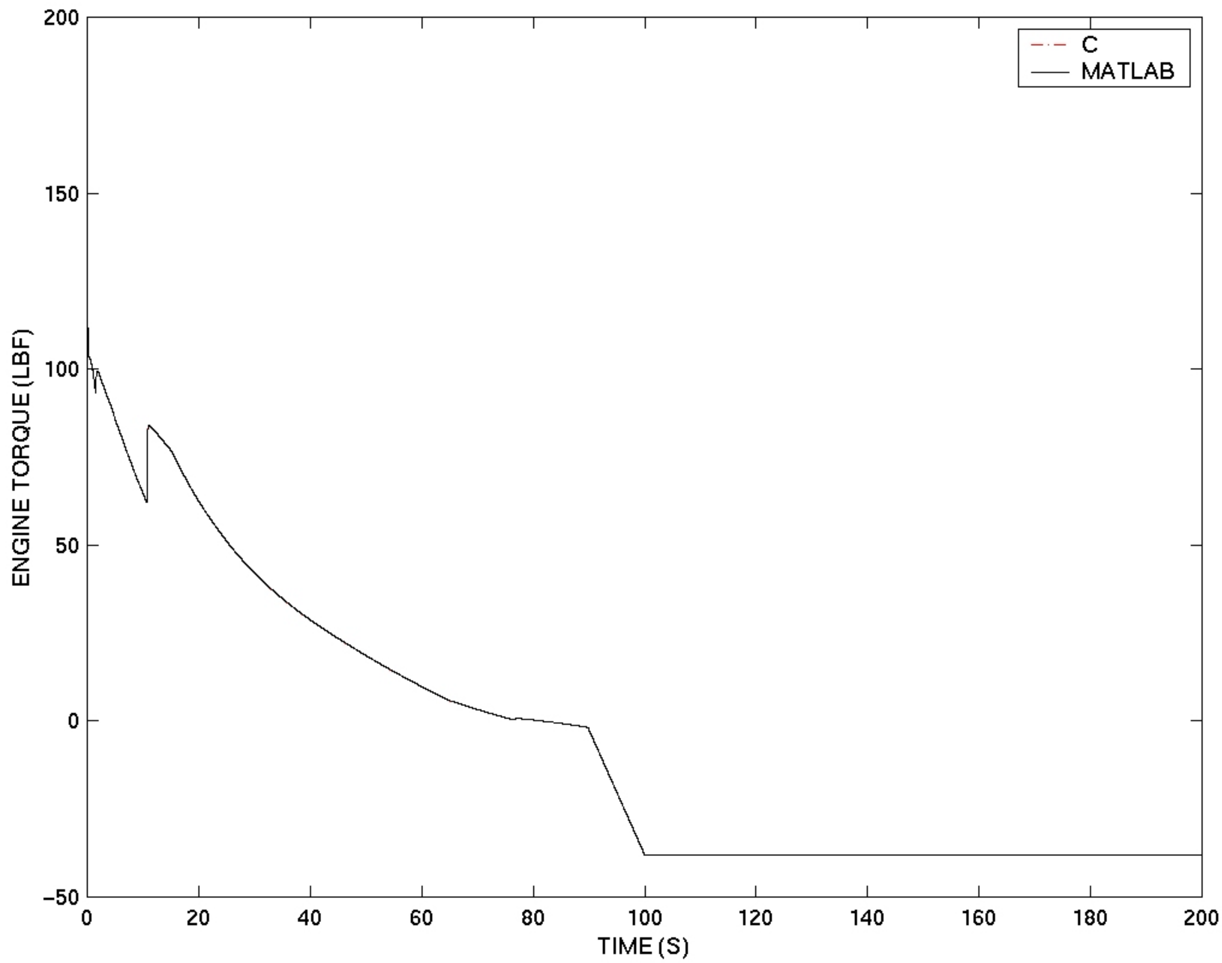
Case 2



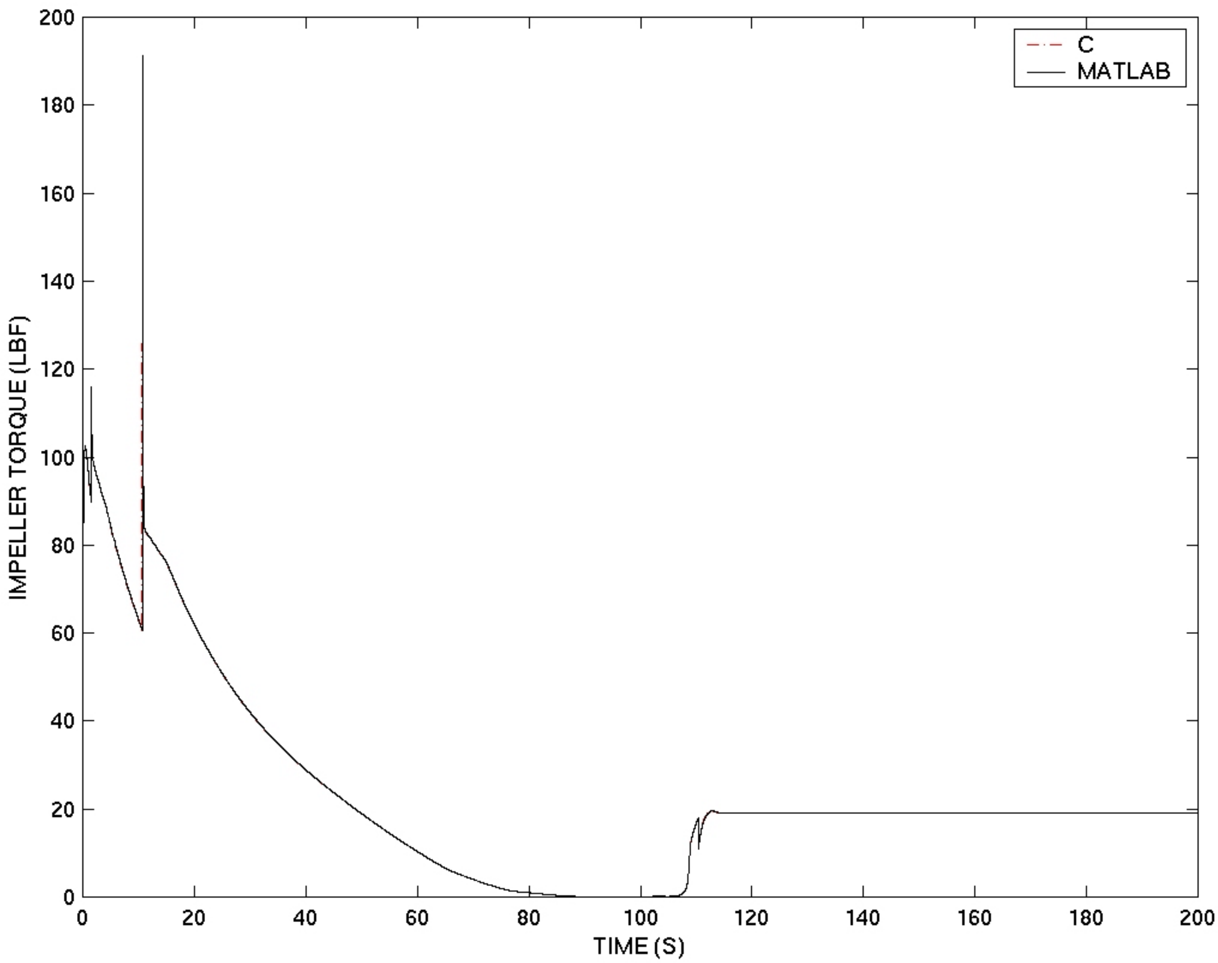
C
MATLAB
THROTTLE (%)
TIME (S)
0
2
4
6
8
10
12
14
16
18
20
0
20
40
60
80
100
120
140
160
180
200


C
MATLAB
BRAKE FORCE (LBF)
TIME (S)
0
50
100
150
200
250
300
350
400
0
20
40
60
80
100
120
140
160
180
200

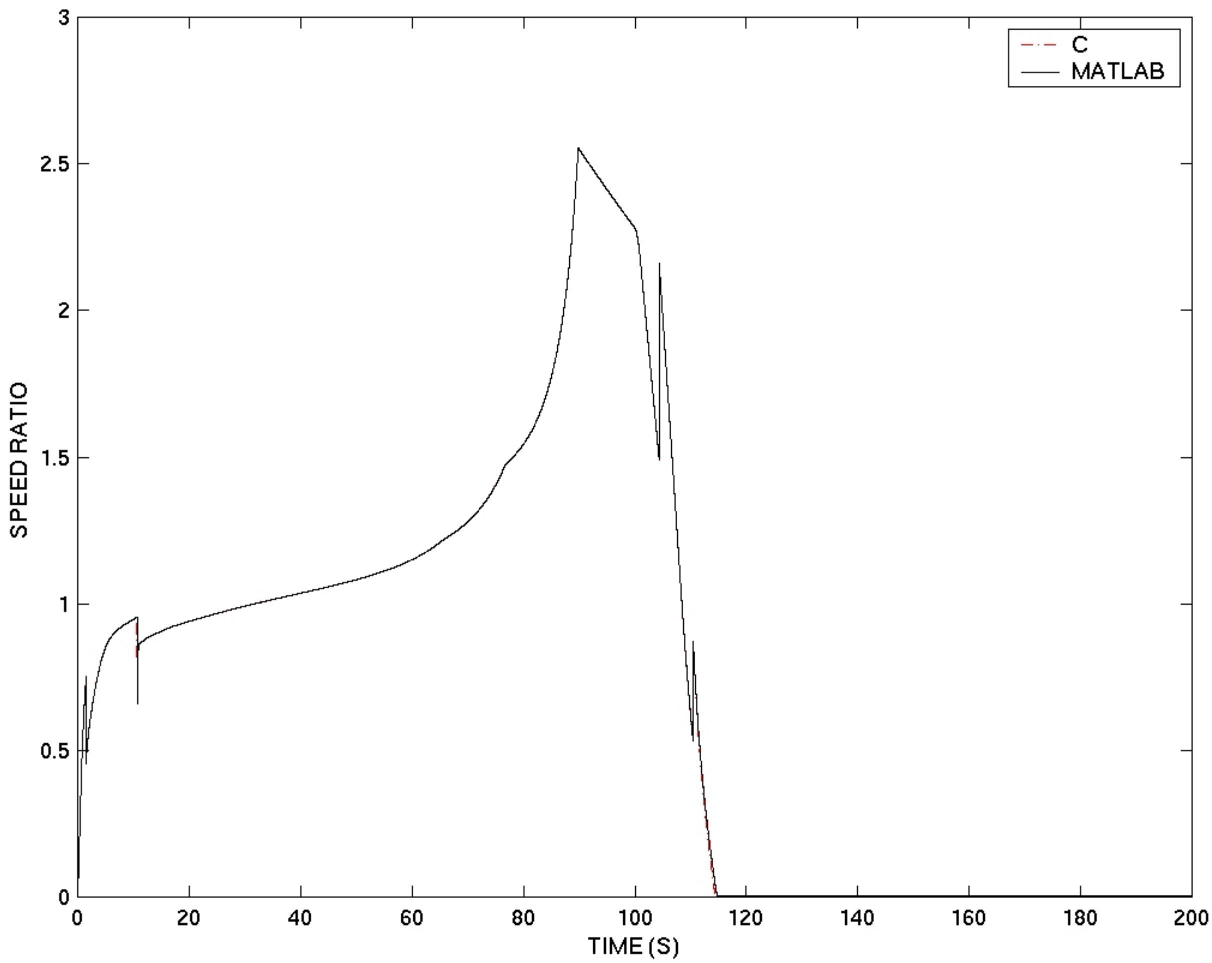

C
MATLAB
ENGINE SPEED (RPM)
TIME (S)
2200
2000
1800
1600
1400
1200
1000
800
600
0
20
40
60
80
100
120
140
160
180
200



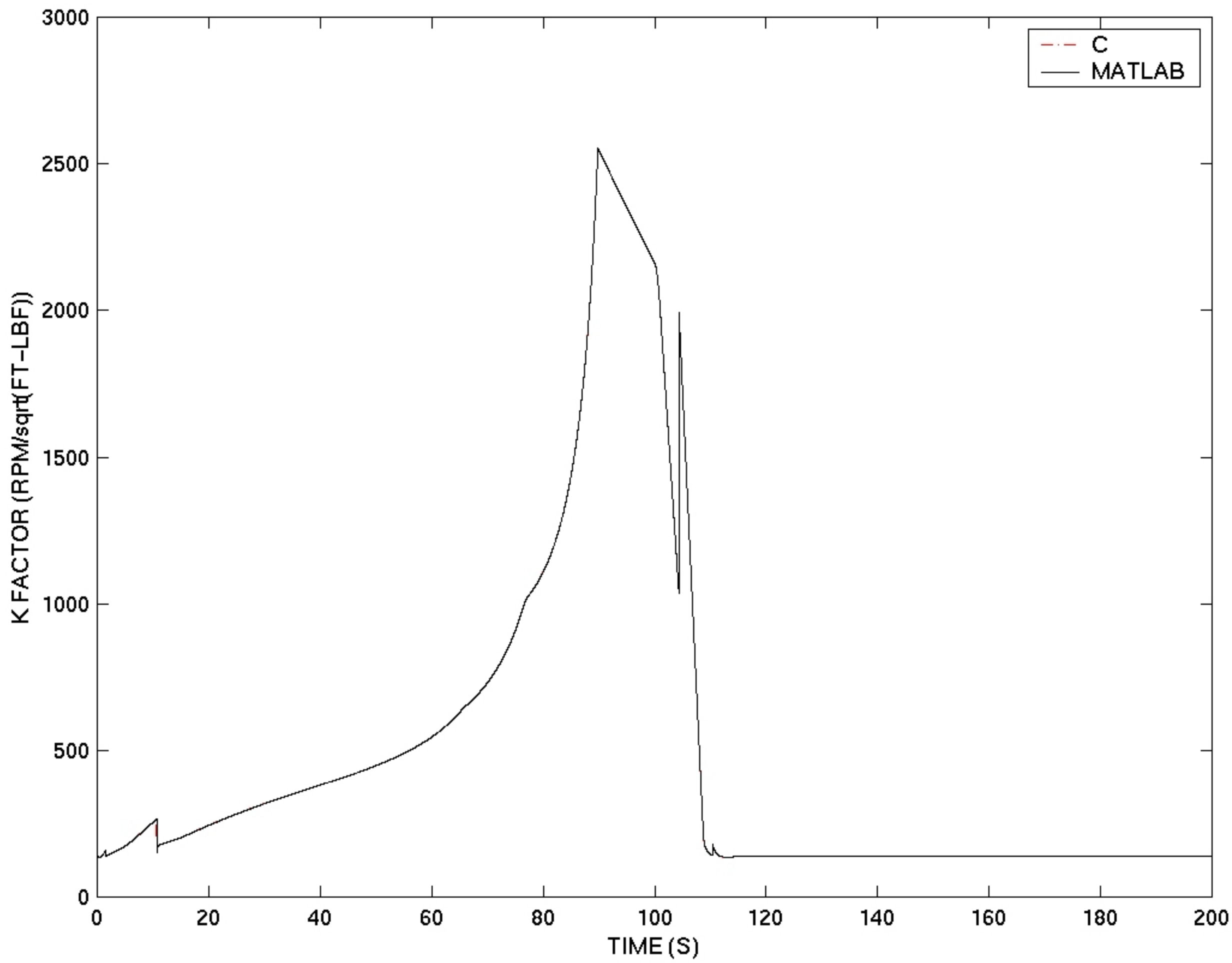
200
150
100
50
0
−50
0
20
40
60
80
100
120
140
160
180
200
ENGINE TORQUE (LBF)
TIME (S)
C
MATLAB



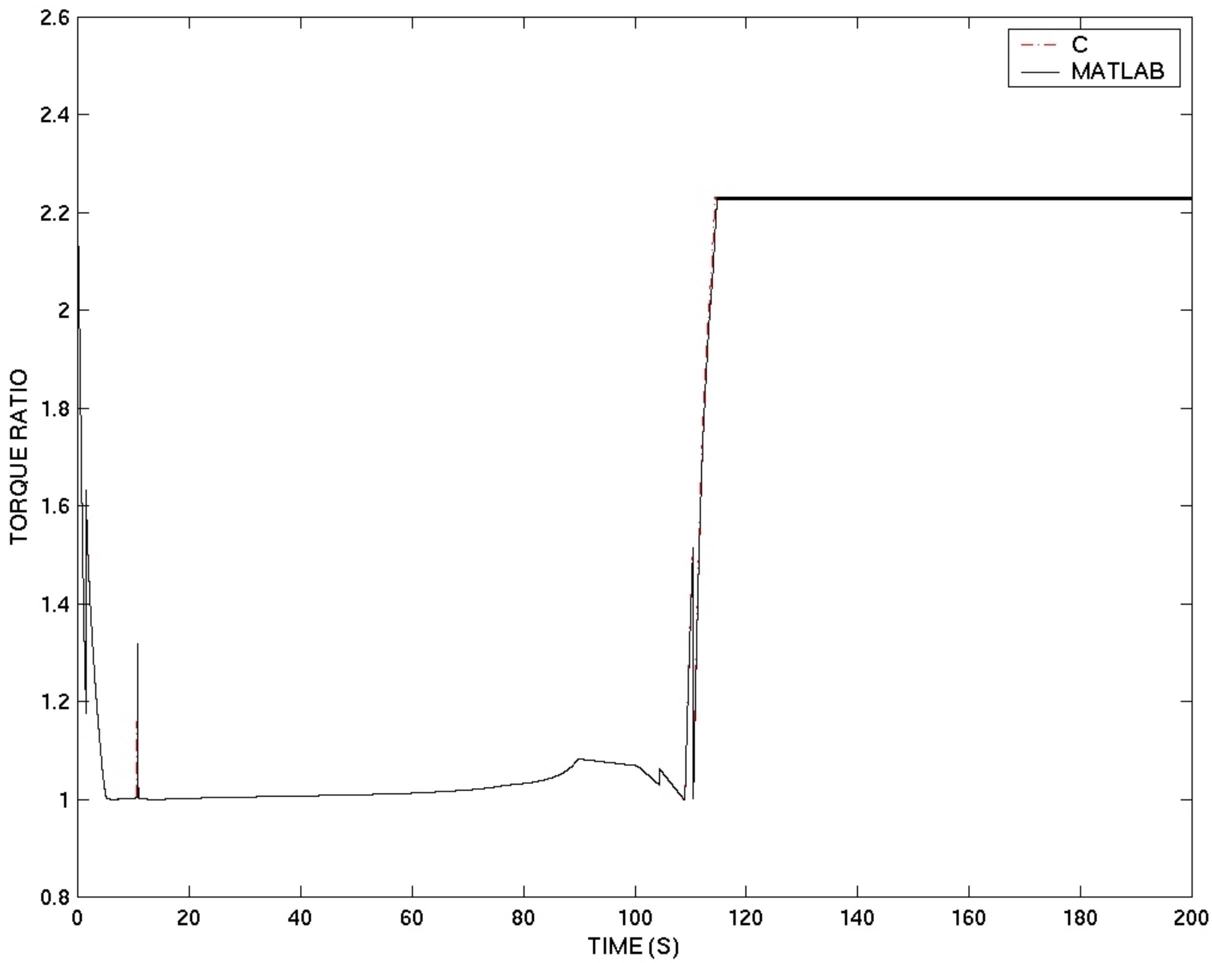
C
MATLAB
IMPELLER TORQUE (LBF)
TIME (S)
0
20
40
60
80
100
120
140
160
180
200

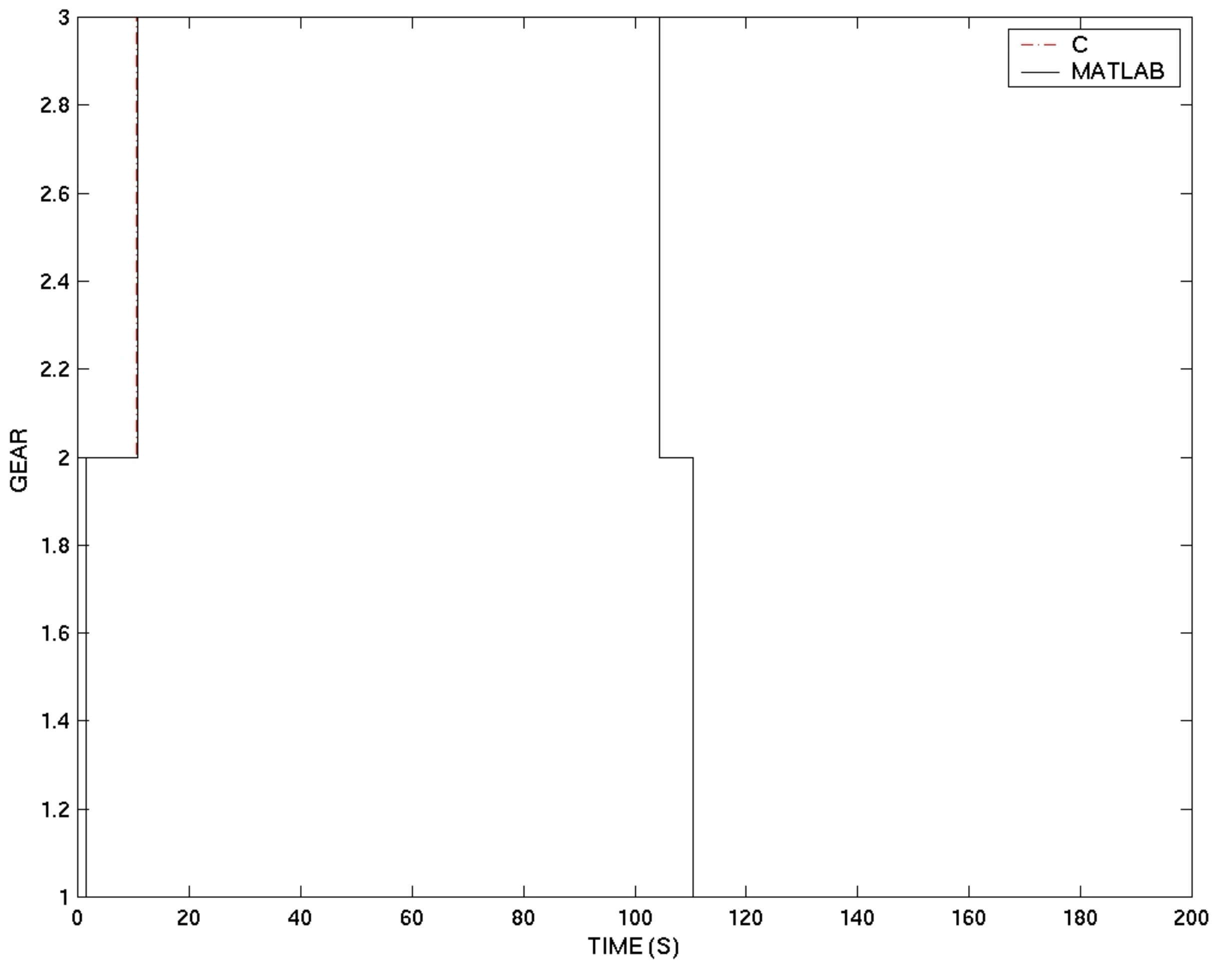


C
MATLAB
SPEED RATIO
TIME (S)
0
0.5
1
1.5
2
2.5
3
0
20
40
60
80
100
120
140
160
180
200

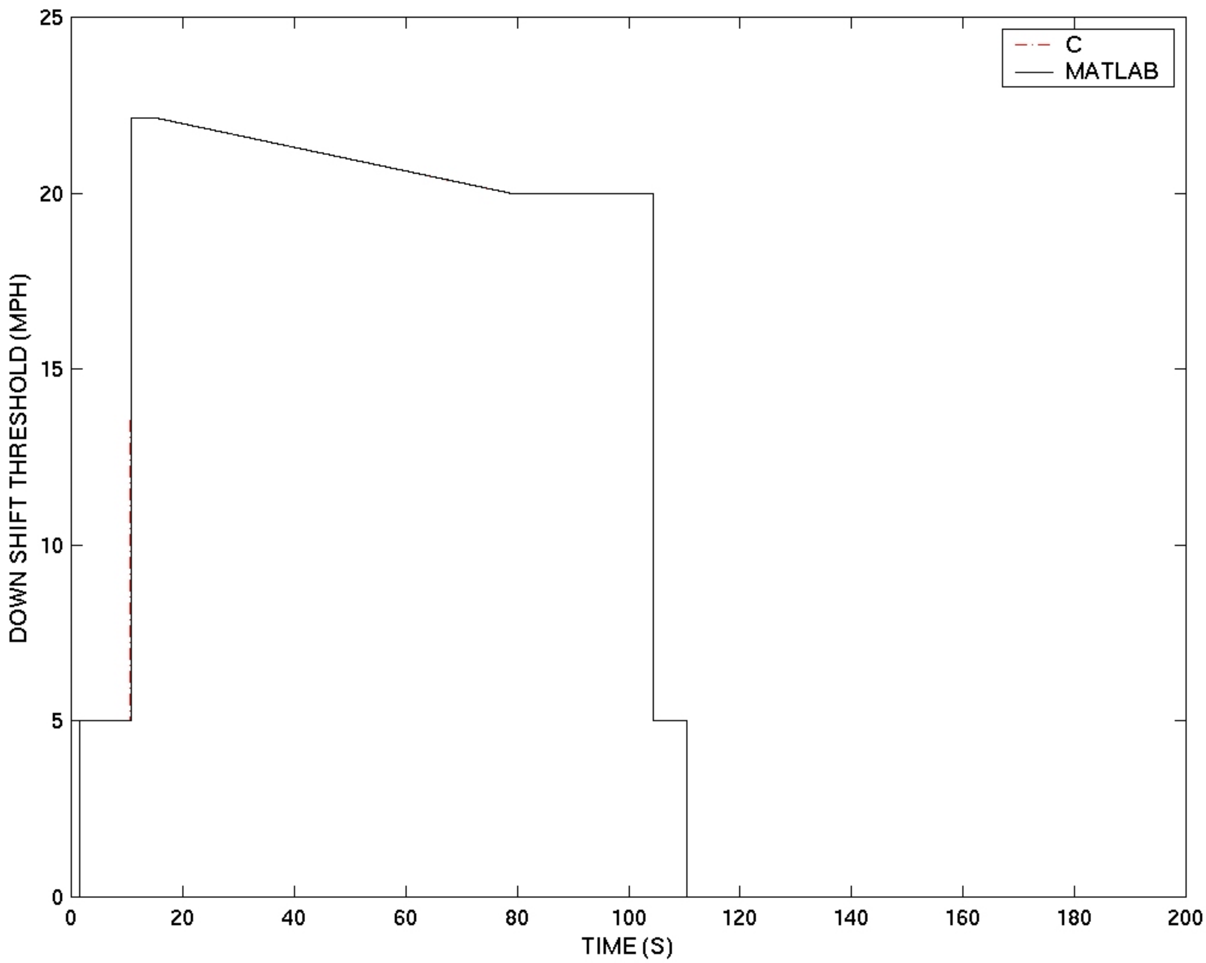


3000
2500
2000
1500
1000
500
0
0
20
40
60
80
100
120
140
160
180
200
K FACTOR (RPM/sqrt(FT-LBF))
TIME (S)
C
MATLAB

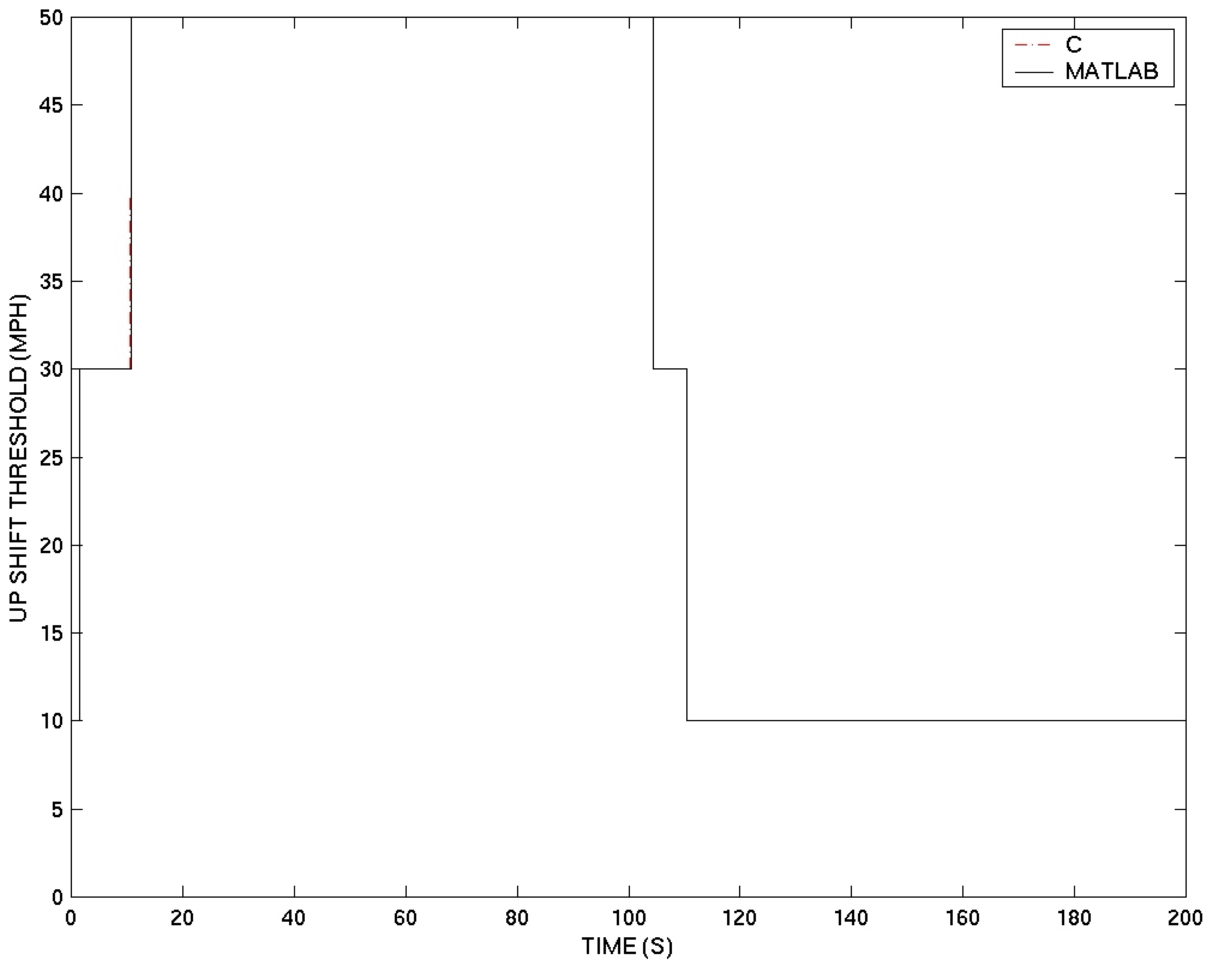

C
MATLAB
TORQUE RATIO
TIME (S)
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0
20
40
60
80
100
120
140
160
180
200



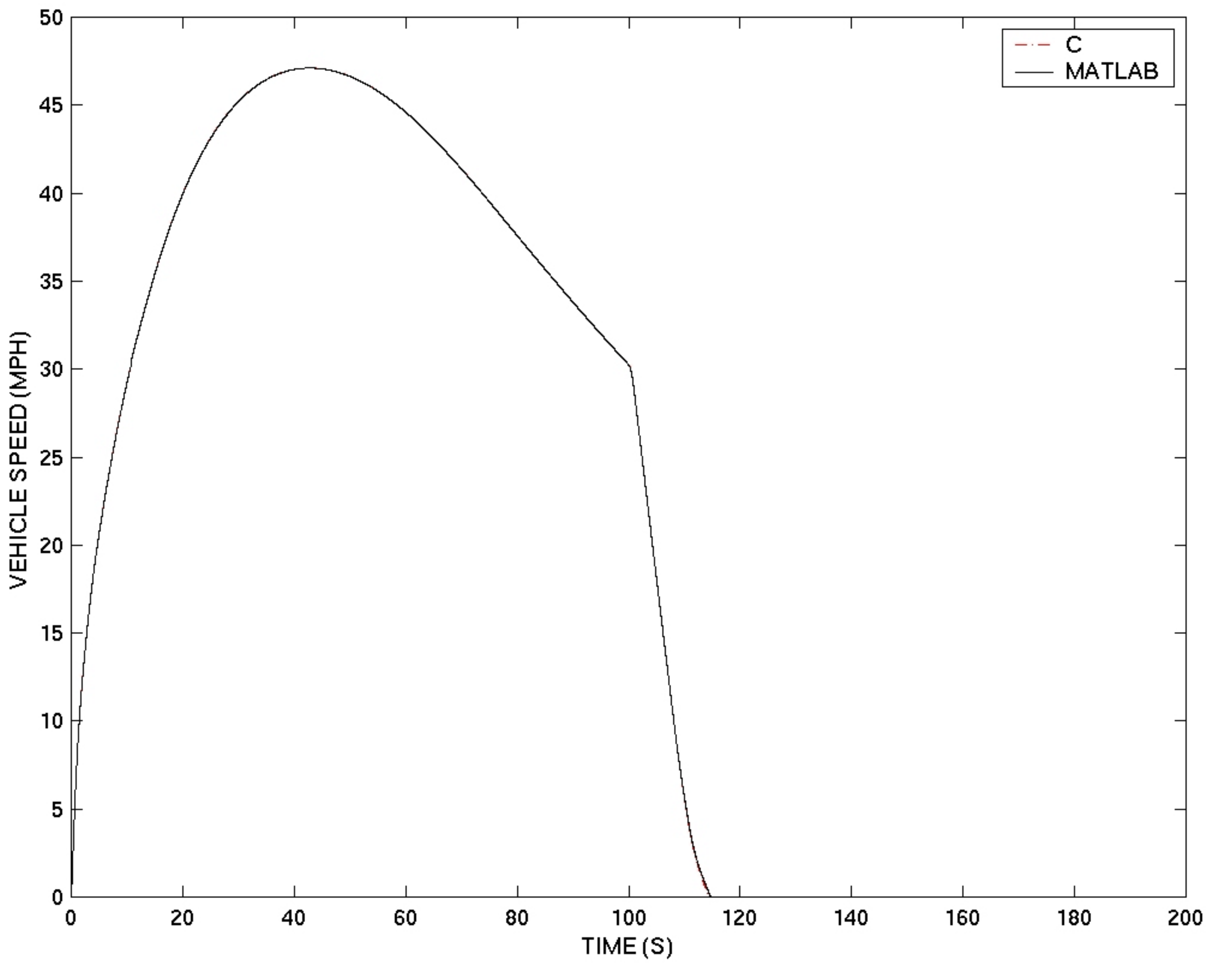
C
MATLAB
GEAR
TIME (S)



25
20
15
10
5
0
0
20
40
60
80
100
120
140
160
180
200
DOWN SHIFT THRESHOLD (MPH)
TIME (S)
C
MATLAB



C
MATLAB
UP SHIFT THRESHOLD (MPH)
TIME (S)
0
5
10
15
20
25
30
35
40
45
50
0
20
40
60
80
100
120
140
160
180
200



C
MATLAB
VEHICLE SPEED (MPH)
TIME (S)
0
5
10
15
20
25
30
35
40
45
50
0
20
40
60
80
100
120
140
160
180
200